QbC: Quantum Correctness by Construction

ANURUDH PEDURI, Ruhr University Bochum, Germany
INA SCHAEFER, Karlsruhe Institute of Technology, Germany
MICHAEL WALTER, Ruhr University Bochum, Germany

Thanks to the rapid progress and growing complexity of quantum algorithms, correctness of quantum programs has become a major concern. Pioneering research over the past years has proposed various approaches to formally verify quantum programs using proof systems such as quantum Hoare logic. All these prior approaches are post-hoc: one first implements a program and only then verifies its correctness. Here we propose *Quantum Correctness by Construction (QbC)*: an approach to constructing quantum programs from their specification in a way that ensures correctness. We use pre- and postconditions to specify program properties, and propose sound and complete refinement rules for constructing programs in a quantum while language from their specification. We validate QbC by constructing quantum programs for idiomatic problems and patterns. We find that the approach naturally suggests how to derive program details, highlighting key design choices along the way. As such, we believe that QbC can play a role in supporting the design and taxonomization of quantum algorithms and software.

CCS Concepts: • **Theory of computation → Logic and verification**; **Hoare logic**; **Quantum computation theory**.

Additional Key Words and Phrases: quantum Hoare logic, correctness by construction, quantum while language

ACM Reference Format:
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

1 Introduction

The field of quantum computing has seen tremendous progress. There are a variety of quantum algorithms for a broad range of computational problems, including for combinatorial search and optimization [4, 6, 16, 30], factoring and other algebraic problems [66], and linear algebra [19, 26, 31, 73] – supported by algorithmic frameworks such as quantum walks [20] and the quantum singular value transform [26, 43], as well as by novel quantum data structures [27, 81]. Montanaro [44] provides an overview of various quantum algorithms. To support these developments and the construction of larger quantum programs, Selinger [64] first proposed a design for a quantum programming language. Since then, there have been numerous quantum programming languages at various levels of abstraction (see, e.g., [3, 12, 28, 69, 71, 74] and references therein).

Correctness of algorithms has always been a major concern in computing, with intensive work on program analysis, testing and verification of programs and software in past decades. For classical computing, Hoare [33] introduced a formal system, in which for a program S, one specifies its properties by using a precondition P and a postcondition Q, resulting in a *Hoare triple* denoted

Authors' Contact Information: Anurudh Peduri, anurudh.peduri@rub.de, Chair for Quantum Information, Faculty of Computer Science, Ruhr University Bochum, Bochum, Germany; Ina Schaefer, ina.schaefer@kit.edu, Chair of Testing, Validation and Analysis of Software-Intensive Systems (TVA), Institute for Information Security and Dependability (KASTEL), Karlsruhe Institute of Technology, Karlsruhe, Germany; Michael Walter, michael.walter@rub.de, Chair for Quantum Information, Faculty of Computer Science, Ruhr University Bochum, Bochum, Germany.

ACM 2475-1421/2025/4-ART99
https://doi.org/10.1145/3720433

as $\{P\}S\{Q\}$. A Hoare triple is said to be correct if running the program S starting in any state satisfying P results in a state that satisfies Q. Hoare logic has been extended to probabilistic programs [46, 46], where the properties are probabilistic, and correctness is defined in terms of their expectations. A survey of the successes of Hoare logic can be found in [7]. Another approach to verifying programs is using Incorrectness Logic [49], which attempts to find bugs in programs by finding counterexamples. Quantum computing poses unique challenges for formal verification that are not encountered in classical computing due to the nature of its computational model. Pioneering research over the past years has uncovered how to adapt the above-mentioned approaches to the quantum setting, e.g., quantum Hoare logic [18, 34, 63, 80] and quantum incorrectness logic [78].

Still, all prior approaches to verifying quantum programs are *post-hoc*: they take the completed program as a starting point and establish whether the program meets the specification. If post-hoc verification fails, there is often no indication of what needs to be fixed in the program. Especially in the quantum setting, predicates specifying program properties are represented by large matrices, which makes it difficult to reason about, locate, and fix issues with quantum programs with a post-hoc approach. In contrast, *Correctness-by-Construction (CbC)* [23, 38, 45] is a programming methodology to incrementally build correct programs based on a specification. For classical computing, CbC provides a method where one starts with a concise specification, and then uses a small set of refinement rules to incrementally construct the program in such a way that at the end of the construction process, the program provably satisfies the specification, i.e., it is correct by construction. This approach can help build well-structured and concise programs and draw insights from the corresponding specification, supporting algorithm designers in developing intuition and allowing them to focus on central design aspects of the constructed algorithms. Runge et al. [62] provide a detailed comparison between CbC and post-hoc verification techniques. To the best of our knowledge, the Correctness-by-Construction approach has not been applied to construct and verify quantum programs before our work.

1.1 Our Contributions

In this work, we propose *Quantum Correctness by Construction (QbC)*, an approach to constructing quantum programs from their specification in a way that ensures correctness. To this end, we consider a simple quantum programming language, the *quantum while language*, and extend it with a new construct called a *hole*, which represents a yet-to-be-constructed program. Holes take the form $\{P\}\square\{Q\}$, where P and Q represent the pre- and postconditions that should be satisfied by the program, as in quantum Hoare logic [80]. For example, the specification for the paradigmatic problem of *searching* a "database" $f\colon \{0,1\}^n \to \{0,1\}$ with success probability p, as famously solved by Grover's algorithm [30], can be succinctly expressed as

$$\{pI\}\square\{\textstyle\sum_{x \text{ s.t. } f(x)=1} |x\rangle\langle x|\}\,.$$

This states that measuring the program's output results in a solution to the search problem, i.e., an x such that $f(x) = 1$, with probability at least p. Indeed, the precondition pI accepts any state with probability p, and the postcondition, which then must hold with at least this probability, accepts only states that on measuring give an x such that $f(x) = 1$. We discuss this example in Section 4.

For the above language, we then provide *refinement rules*, which allow filling in holes in quantum programs in such a way that correctness is preserved. There are two widely used notions of correctness for Hoare triples, *partial* and *total* correctness, which differ in how they treat non-termination. We provide refinement rules for both notions. These rules take the following form:

$$\{P\}\square\{Q\} \hookrightarrow S \qquad \text{if certain conditions } C_1, C_2, \ldots \text{ hold}$$

To apply such a rule, one first checks that the conditions $C_1, C_2, \ldots$ are satisfied and then replaces the left-hand side hole with the right-hand side program S (which may itself contain other holes). For example, we can always apply the *sequence rule*, $\{P\}\square\{Q\} \hookrightarrow \{P\}\square\{R\};\{R\}\square\{Q\}$; intuitively, this rule states that in order to construct a program that takes precondition P to postcondition Q, it suffices to construct a program that takes the precondition to some intermediate condition R, and another one that takes this intermediate condition to the postcondition. Starting from an initial specification $\{P\}\square\{Q\}$, one iteratively applies refinement rules until one arrives at a program S_{final} that contains no more holes. We prove a *soundness theorem* that states that any program obtained by this process of refinement is guaranteed to satisfy the initial specification, that is, the quantum Hoare triple $\{P\}S_{\text{final}}\{Q\}$ is valid and the program is correct by construction. Here, we use the notion of quantum Hoare triple introduced by Ying [80]. We also prove *completeness*: any program that satisfies a specification can be obtained by process of refinement. We note that the completeness above is *relative* to the theory of complex numbers, as defined in Ref. [80], which is also assumed in all prior works. Because of the unintuitive nature of quantum logic, these results are more challenging to establish than in the classical case. For example, quantum predicates cannot be interpreted as (deterministic or probabilistic) functions of some program state; observing or "measuring" predicates is not a passive operation, but will in general change the program's state; and there is in general no canonical choice of predicates in refinement rules. All these can be traced back to the noncommutative nature of quantum information.

Finally, we validate our approach by constructing quantum programs for *quantum teleportation* and *quantum search* (Section 4). In each case, we start from their intuitive specification and use one key algorithmic idea or refinement step at a time. We find that the refinement rules not only guide the construction of the desired programs, but that the QbC approach also reveals design choices that can be made along the way. For example, when constructing a program for the quantum search problem, we show how one can naturally arrive at both a naive algorithm that proceeds by random sampling, as well as Grover's celebrated search algorithm that offers a quantum speedup. To summarize, in this work, we:

(1) Introduce a Quantum Correctness by Construction (QbC) approach for quantum programs. In QbC, quantum programs are constructed starting from specifications by successively applying refinement rules.
(2) In doing so, provide a formalization based on quantum while programs with *holes*, which specify pre- and postconditions of subprograms that still need to be constructed.
(3) Prove our refinement systems sound and complete: Any program constructed from an initial specification by using the refinement rules must satisfy the specification, and for any program satisfying a specification, there exists a sequence of refinements to obtain it from the specification.
(4) Validate QbC by constructing quantum programs for idiomatic problems, starting from their specification. Our findings suggest that QbC can play a role in supporting the design and taxonomization of quantum algorithms and software.

1.2 Related Work

Correctness-by-Construction for Classical Programs. The CbC approach to programming was initially introduced by Back et al. [9], Dijkstra [23], Morgan [45] and Gries [29] and later extended by Kourie and Watson [38], see also the relation algebra of programming [13]. There has since been extensive work on tools and theory for CbC [14, 15, 37, 58–61]. Other similar approaches include the B method [1, 2, 17], syntax-guided synthesis [68], and type systems [21].

Probabilistic Verification. Various classical program logics have been extended to probabilistic programs: Hoare logic [22], weakest precondition reasoning [46, 50], separation logic [11], relational logic [8, 10], and dynamic logic for forward reasoning [53]. We refer to Kaminski et al. [35] and references therein for recent work on proving termination and bounding expected runtimes. Syntax-guided synthesis techniques have also been extended to probabilistic programs [5].

Quantum Verification. The majority of work in quantum verification has focused on post-hoc approaches. The first approach on Hoare logic for quantum programs was introduced by [18, 34], and was extended to support reasoning about unbounded quantum loops with (relative) completeness [80] and to include classical variables [24]. Zhou et al. [82] provided a rigorous implementation of quantum Hoare logic in Coq with a range of applications. Zhou et al. [83] focused on projections as predicates and proposed a notion of robust Hoare triples, which allow pre- and postconditions to be approximately satisfied. Rand [55] surveys recent advances in Hoare-style verification logics for quantum programs. Zuliani [84] illustrates a refinement based approach, but gives no complete system of refinement rules. Neri et al. [47] contributes an extension of laws of classical program algebra to quantum programming. Other related quantum verification efforts include incorrectness logic [78], circuit verification [39, 54, 56], quantum relational Hoare logic [40, 72], and equational reasoning in Dirac notation [77].

Recent Developments. Shortly after our work had appeared as a preprint, Feng et al. [25] reported on independent work on a refinement system that is similar to ours but differs in two key respects. First, their work considers only projective predicates, whereas our formalism allows for arbitrary predicates. Therefore our formalism captures a larger class of interesting properties, in particular success probabilities of algorithms, which are important in most quantum algorithms (see Section 4 and Section 4.4). Second, while their refinement system only ensures partial correctness, we also provide a refinement system that ensures total correctness. The latter gives stronger guarantees and in particular enables reasoning about termination, which is impossible otherwise. Feng et al. [25] also provide a Python-based proof-of-concept implementation of their calculus. A recent work [65] has similarly proposed a web-based proof-of-concept implementation of QbC.

Organization of the Paper. In Section 2, we review the basic quantum formalism that is used in the paper. We introduce the *quantum while language*, a simple quantum programming language with control-flow and loops, as well as *quantum Hoare logic*, which defines notions of correctness of quantum programs in terms of pre- and postconditions. In Section 3, we introduce *Quantum Correctness by Construction (QbC)*. We first define an extension of the quantum while language, called *programs with holes*, which allows specifying subprograms that still need to be constructed. Then we define refinement rules that can be used to construct quantum programs from given specifications, and we prove soundness and completeness of these rules. In Section 4, we use QbC to naturally construct several quantum programs starting from their specification. We conclude in Section 5. Appendices A to C contain technical proofs for results announced in the main text. Appendix D continues the discussion of a running example in the text.

2 Preliminaries

In this section, after setting our notation and conventions (Section 2.1), we give a brief introduction to the formalism of quantum computing (Section 2.2). Then we describe the syntax and semantics of a simple quantum programming language (Section 2.3) and recall quantum Hoare logic (Section 2.4).

2.1 Notation and Conventions

We take $\mathbb{N}$ to be the set of natural numbers including zero. In this work, a Hilbert space $\mathcal{H}$ is a

finite-dimensional complex vector space with inner product. Throughout the paper we use *Dirac notation*: we write $|\psi\rangle \in \mathcal{H}$ for vectors, $\langle\phi|$ for covectors, and $\langle\phi|\psi\rangle$ for the inner product. Here, ψ is an arbitrary label. In general, $M^\dagger$ denotes the adjoint of a linear operator M. The identity operator on a Hilbert space $\mathcal{H}$ is denoted by $I_\mathcal{H}$ and can be written as $I_\mathcal{H} = \sum_{x\in\Sigma} |x\rangle\langle x|$ for any (orthonormal) basis $\{|x\rangle\}_{x\in\Sigma}$ of $\mathcal{H}$, where Σ is an index set. We write I when the Hilbert space is clear from the context. The *trace* of an operator M can be computed as $\operatorname{tr} M = \sum_{x\in\Sigma} \langle x|M|x\rangle$ for any basis as above. For example, a *quantum bit* or *qubit* corresponds to the 2-dimensional Hilbert space $\mathcal{H} = \mathbb{C}^2$, with standard basis $\{|0\rangle, |1\rangle\}$ labeled by $\Sigma = \{0, 1\}$. The identity operator is $I = |0\rangle\langle 0| + |1\rangle\langle 1|$. An example of an operator on $\mathbb{C}^2$ is the Pauli X matrix, defined as $X = |0\rangle\langle 1| + |1\rangle\langle 0|$. It satisfies $X^\dagger = X$ and $\operatorname{tr} X = 0$. We require two more concepts from linear algebra. An operator M on $\mathcal{H}$ is *Hermitian* if $\langle\psi|M|\psi\rangle \in \mathbb{R}$ for all vectors $|\psi\rangle \in \mathcal{H}$, and *positive semidefinite (PSD)* if $\langle\psi|M|\psi\rangle \geq 0$ for all vectors $|\psi\rangle \in \mathcal{H}$. Equivalently, M can be diagonalized by a unitary matrix and has real resp. nonnegative eigenvalues. Given two operators A and B on $\mathcal{H}$, we write $A \preceq B$ if and only if $B - A$ is PSD; this defines the *Löwner order*. For example, we can write $M \geq 0$ to state that M is PSD.

2.2 Quantum Computing

We now recall the basic formalism of quantum computing. We refer to the excellent textbooks Nielsen and Chuang [48], Wilde [76], Yanofsky and Mannucci [79] for more comprehensive introductions.

Variables. A *quantum variable* q is modeled by a Hilbert space $\mathcal{H}_q = \mathbb{C}^{\Sigma_q}$ for some finite index set Σ_q. This means that $\mathcal{H}_q$ is a vector space equipped with an inner product and an orthonormal *standard basis* (or *computational basis*) $\{|x\rangle\}_{x\in\Sigma_q}$, labeled by the elements $x \in \Sigma_q$. When $\Sigma_q = \{0, 1\}$, then $\mathcal{H}_q = \mathbb{C}^2$ and q is called a *quantum bit* or *qubit*, with standard basis $\{|0\rangle, |1\rangle\}$, as above. If $\boldsymbol{q}$ is a collection of quantum variables, then the corresponding Hilbert space is $\mathcal{H}_{\boldsymbol{q}} = \bigotimes_{q\in\boldsymbol{q}} \mathcal{H}_q \cong \mathbb{C}^{\Sigma_{\boldsymbol{q}}}$, where the Cartesian product $\Sigma_{\boldsymbol{q}} = \prod_{q\in\boldsymbol{q}} \Sigma_q$ labels the standard (product) basis of the quantum variables $\boldsymbol{q}$. We assume that there is a finite set of quantum variables, denoted by $\mathbf{qVars}$. Then the overall Hilbert space is

$$\mathcal{H} = \mathcal{H}_{\mathbf{qVars}} = \bigotimes_{q\in\mathbf{qVars}} \mathcal{H}_q \cong \bigotimes_{q\in\mathbf{qVars}} \mathbb{C}^{\Sigma_q} = \mathbb{C}^\Sigma,$$

where $\Sigma = \Sigma_{\mathbf{qVars}} = \prod_{q\in\mathbf{qVars}} \Sigma_q$ labels the standard basis of the all quantum variables. It is well understood how to extend the above to infinite-dimensional Hilbert spaces and an infinite number of quantum variables, but we will not need this here.

States. The *state* of all the quantum variables is described by a positive semidefinite (PSD) operator on $\mathcal{H}$ with trace equal to 1, often called a *density operator*. We denote the set of all such operators by $\mathcal{D}(\mathcal{H})$. A state is called *pure* if it is given by a rank-one projection, i.e., if $\rho = |\Psi\rangle\langle\Psi|$ for some unit vector $|\Psi\rangle \in \mathcal{H}$. For simplicity one often also refers to $|\Psi\rangle$ as the (pure) state. States that are not pure are called *mixed*. More generally, we consider *partial states*, also called *subnormalized states*, which are PSD operators of trace at most one, denoted by $\mathcal{D}_{\leq 1}(\mathcal{H})$. Partial states are akin to sub-probability measures in probabilistic computing. They are useful for reasoning about programs that may terminate with probability less than one. For example, the computational basis states of a qubit are $|0\rangle\langle 0|$ and $|1\rangle\langle 1|$, which are both pure states, and the partial state $\rho = 0.8\,|0\rangle\langle 0|$ describes the output of a program terminating in state $|0\rangle\langle 0|$ with probability 0.8 (and not terminating otherwise).

Operations. There are two basic kinds of operations. The first is to apply a *unitary*. An operator U is called a *unitary* if $UU^\dagger = U^\dagger U = I$. If we apply a unitary U on $\mathcal{H}$ to a (partial) state ρ, the result is $U\rho U^\dagger$, which is again a (partial) state. For example, the Hadamard matrix $H = \frac{1}{\sqrt{2}}\left(\begin{smallmatrix} 1 & 1 \\ 1 & -1 \end{smallmatrix}\right)$ is a one-qubit unitary, and on applying it to the input state $|0\rangle\langle 0|$, we get $|+\rangle\langle +|$, where $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$.

The second operation is to measure the quantum state. A *measurement* (also called a *positive operator-valued measure* or *POVM*) is given by a family of positive semidefinite operators $\boldsymbol{M} = \{M_\omega\}_{\omega\in\Omega}$, labeled by some finite index set Ω, such that $\sum_{\omega\in\Omega} M_\omega = I$. If one measures a (partial) state ρ then the probability of seeing outcome $\omega \in \Omega$ is $q_\omega = \mathrm{tr}(\rho M_\omega)$, in which case the state changes to $\rho_\omega = \sqrt{M_\omega}\rho\sqrt{M_\omega}/q_\omega$. Note that any PSD operator M has a unique PSD *square root*, denoted by $\sqrt{M}$. If M is a projection, meaning $M^2 = M$, then $\sqrt{M} = M$. We abbreviate $\mathcal{M}_\omega(\rho) = \sqrt{M_\omega}\rho\sqrt{M_\omega}$. Note that this is a partial state, with trace equal to the probability q_ω of outcome ω.

When $\Omega = \{0, 1\}$, there are only two possible outcomes, and this is called a *binary measurement*. Any binary measurement $\{B_0, B_1\}$ can be obtained by picking a positive semidefinite operator B with $B \preceq I$ and setting $B_0 = I - B$ and $B_1 = B$. As above, we abbreviate $\mathcal{B}_0(\rho) = \sqrt{I-B}\rho\sqrt{I-B}$ and $\mathcal{B}_1(\rho) = \sqrt{B}\rho\sqrt{B}$. For example, for a qubit, $B = |1\rangle\langle 1|$ defines the standard basis measurement, with $B_0 = \sqrt{B_0} = |0\rangle\langle 0|$ and $B_1 = \sqrt{B_1} = |1\rangle\langle 1|$. For succinctness, we will often refer to B rather than $\{B_0, B_1\}$ as a binary measurement. For example, if we apply the binary standard basis measurement to a qubit in the $|+\rangle\langle +|$ state, then $\mathcal{B}_0(|+\rangle\langle +|) = \frac{1}{2}|0\rangle\langle 0|$ and $\mathcal{B}_1(|+\rangle\langle +|) = \frac{1}{2}|1\rangle\langle 1|$. Thus each outcome $\omega \in \{0, 1\}$ occurs with probability half and the state after the measurement is $|\omega\rangle\langle\omega|$.

We can also apply any of the above operations to a subset $\boldsymbol{q} \subseteq \mathbf{qVars}$ of the quantum variables. To this end, let us, for an arbitrary operator A on $\mathcal{H}_{\boldsymbol{q}}$, define the operator $A_{\boldsymbol{q}} = A \otimes I$ on $\mathcal{H}$, where the tensor product is with respect to the decomposition $\mathcal{H} = \mathcal{H}_{\boldsymbol{q}} \otimes \mathcal{H}_{\boldsymbol{q}^c}$ and $\boldsymbol{q}^c = \mathbf{qVars} \setminus \boldsymbol{q}$ denotes the remaining quantum variables. In prior work, this has been called *cylindrical extension* [82]. Then, if U is a unitary on $\mathcal{H}_{\boldsymbol{q}}$, we can apply it to the quantum variables $\boldsymbol{q}$ by taking $U_{\boldsymbol{q}}$ in the above prescription. Similarly, if $\{M_\omega\}_{\omega\in\Omega}$ is a measurement on $\mathcal{H}_{\boldsymbol{q}}$, we take $\{M_{\omega,\boldsymbol{q}}\}$ in the above prescription; we also abbreviate $\mathcal{M}_{\omega,\boldsymbol{q}}(\rho) = \sqrt{M_{\omega,\boldsymbol{q}}}\rho\sqrt{M_{\omega,\boldsymbol{q}}}$.

2.3 Quantum While Language

In this paper, we consider a *quantum while language* [24, 80, 82]. We first introduce its syntax and then discuss its denotational semantics. The language supports initializing quantum variables, applying unitary operations, and classical control flow based on quantum measurement outcomes. We first describe the language's syntax.

Definition 2.1 (Syntax). Quantum while programs are given by the following grammar:

$$\begin{aligned} S \;::=\;& \mathbf{skip} \mid \boldsymbol{q} := |\mathbf{0}\rangle \mid \boldsymbol{q} := U(\boldsymbol{q}) \mid S_1; S_2 \mid \mathbf{repeat}\ N\ \mathbf{do}\ S\ \mathbf{end} \\ & \mid \mathbf{case\ meas}\ \boldsymbol{q}\ \mathbf{with}\ \{\omega_1 : M_{\omega_1}, \omega_2 : M_{\omega_2}, \ldots\}\ \mathbf{of}\ \omega_1 : S_{\omega_1},\ \omega_2 : S_{\omega_2},\ \ldots\ \mathbf{end} \\ & \mid \mathbf{while\ meas}\ \boldsymbol{q}\ \mathbf{with}\ B\ \mathbf{do}\ S\ \mathbf{end} \end{aligned}$$

where $S, S_1, S_2, S_{\omega_1}, S_{\omega_2}, \ldots$ denote programs in the language; $\boldsymbol{q}$ denotes a set of quantum variables; U is a unitary operator on $\mathcal{H}_{\boldsymbol{q}}$; N is a natural number; $\{M_\omega\}_{\omega\in\Omega}$ is a measurement on $\mathcal{H}_{\boldsymbol{q}}$ with outcomes in some set $\Omega = \{\omega_1, \omega_2, \ldots\}$; B is an operator defining a binary measurement $\{I - B, B\}$.

The first five instructions are self-explanatory: $\mathbf{skip}$ is a no-op; $\boldsymbol{q} := |\mathbf{0}\rangle$ initializes the set of variables $\boldsymbol{q}$; $\boldsymbol{q} := U(\boldsymbol{q})$ applies a unitary U to the set of variables $\boldsymbol{q}$; $S_1\ ;\ S_2$ sequentially composes the two programs, running S_1 followed by S_2; and $\mathbf{repeat}\ N\ \mathbf{do}\ S\ \mathbf{end}$ runs the program S N times. The **case** statement is used for control flow: it measures the set of variables $\boldsymbol{q}$ with the measurement $\boldsymbol{M}$, and on seeing outcome ω runs the program S_ω. The **while** statement measures the set of variables $\boldsymbol{q}$ with a binary measurement B and, if it succeeds (i.e. has outcome 1), runs the loop body S and repeats. Thus, both **case** and **while** are classical control flow based on the outcome of a quantum measurement.

Remark 2.2. We comment on an aspect that is well-understood but usually left implicit. In the instruction $\boldsymbol{q} := U(\boldsymbol{q})$, the unitary U will often be restricted to an arbitrary fixed subset of allowed

"gates" (typical choices are few-qubit unitaries or the Clifford+T gate set). The same applies to measurement operators M_ω (typical choices are standard basis measurements or arbitrary 1-qubit measurements). The results of our paper, just like those of prior works, do not depend on this choice (Theorems 3.6 to 3.9), and all our examples use efficiently implementable unitaries.

We also introduce some syntactic sugar for convenience. First, we define an **if** statement as a shorthand for a **case** statement with a binary measurement:

$$\textbf{if meas } \boldsymbol{q} \textbf{ with } B \textbf{ then } S_1 \textbf{ else } S_0 \textbf{ end} \equiv$$
$$\textbf{case meas } \boldsymbol{q} \textbf{ with } \{1\colon B, 0\colon I - B\} \textbf{ of } 1\colon S_1,\ 0\colon S_0 \textbf{ end}$$

We allow leaving out the **else** branch, in which case we take $S_0 =$ **skip**. Lastly, we allow leaving out the measurement in **if**, **case**, and **while**, in which case we default to standard basis measurements:

$$\begin{aligned}
\textbf{if meas } q \textbf{ then } S_1 \textbf{ else } S_0 \textbf{ end} &\equiv \textbf{if meas } q \textbf{ with } |1\rangle\langle 1| \textbf{ then } S_1 \textbf{ else } S_0 \textbf{ end} \\
\textbf{case meas } \boldsymbol{q} \textbf{ of } \ldots \textbf{ end} &\equiv \textbf{case meas } \boldsymbol{q} \textbf{ with } \{x\colon |x\rangle\langle x|\}_{x\in\Sigma_{\boldsymbol{q}}} \textbf{ of } \ldots \textbf{ end} \\
\textbf{while meas } q \textbf{ do } S \textbf{ end} &\equiv \textbf{while meas } q \textbf{ with } |1\rangle\langle 1| \textbf{ do } S \textbf{ end}
\end{aligned}$$

Example 2.3 (Quantum coin toss until zero). As a gentle introduction to the quantum while language, we consider the following program, which exercises several language features:

$$A_{\text{toss-until-zero}} = q := |0\rangle\,; q := H(q); \textbf{while meas } q \textbf{ do } q := H(q) \textbf{ end}$$

It first initializes a qubit in the $|0\rangle$ state, and then repeatedly applies the Hadamard gate and measures in the standard basis, until the outcome "0" is seen. We revisit this example later from the perspectives of denotational semantics and quantum Hoare logic.

Semantics. Any program S defines a function that maps the state of the quantum variables before execution to the corresponding (partial) state after program execution. This is called the *denotational semantics* of S and is defined formally in the following.

Definition 2.4 (Denotational semantics). For any program S, its *denotational semantics* is the map

$$[\![S]\!]\colon \mathcal{D}_{\leq 1}(\mathcal{H}) \to \mathcal{D}_{\leq 1}(\mathcal{H}),$$

which is defined recursively in the following way:

(1) $[\![\ \textbf{skip}\ ]\!](\rho) = \rho$
(2) $[\![\ \boldsymbol{q} := |\boldsymbol{0}\rangle\ ]\!](\rho) = \sum_{\boldsymbol{x}\in\Sigma_{\boldsymbol{q}}} |\boldsymbol{0}\rangle\langle\boldsymbol{x}|_{\boldsymbol{q}}\, \rho\, |\boldsymbol{x}\rangle\langle\boldsymbol{0}|_{\boldsymbol{q}}$
(3) $[\![\ \boldsymbol{q} := U(\boldsymbol{q})\ ]\!](\rho) = U_{\boldsymbol{q}}\rho U_{\boldsymbol{q}}^\dagger$
(4) $[\![\ S_1\,; S_2\ ]\!](\rho) = [\![S_2]\!]([\![S_1]\!](\rho))$
(5) $[\![\ \textbf{repeat } N \textbf{ do } S \textbf{ end}\ ]\!](\rho) = [\![S]\!]^N(\rho)$
(6) $[\![\ \textbf{case meas } \boldsymbol{q} \textbf{ with } \boldsymbol{M} \textbf{ of } \ldots, \omega\colon S_\omega, \ldots \textbf{ end}\ ]\!](\rho) = \sum_{\omega\in\Omega} [\![S_\omega]\!](\mathcal{M}_{\omega,\boldsymbol{q}}(\rho))$
(7) $[\![\ \textbf{while meas } \boldsymbol{q} \textbf{ with } B \textbf{ do } S \textbf{ end}\ ]\!](\rho) = \sum_{k=0}^{\infty}\left(\mathcal{B}_{0,\boldsymbol{q}} \circ \left([\![S]\!] \circ \mathcal{B}_{1,\boldsymbol{q}}\right)^k\right)(\rho)$

We explain the semantics above using the definitions from Section 2.2. Statements (1)-(5) are self-explanatory. The **case** statement (6) measures a set of variables $\boldsymbol{q}$ with the measurement $\boldsymbol{M}$, and on seeing outcome ω executes program S_ω. The **while** statement (7) runs a loop with body S, guarded by the condition that measuring $\boldsymbol{q}$ with the binary measurement B gives outcome 1. The semantics for **while** is well-defined because the partial states $\sigma_n = \sum_{k=0}^{n}\left(\mathcal{B}_{0,\boldsymbol{q}} \circ \left([\![S]\!] \circ \mathcal{B}_{1,\boldsymbol{q}}\right)^k\right)(\rho)$ form an increasing sequence ($\sigma_n \preceq \sigma_{n+1}$ for all n) that is bounded from above. Note that σ_n

represents the partial state corresponding to termination within at most n iterations of the loop. Loops $S =$ **while meas q with B do C end** satisfy the following natural recurrence:

$$[\![S]\!] = \mathcal{B}_{0,\boldsymbol{q}} + [\![S]\!] \circ [\![C]\!] \circ \mathcal{B}_{1,\boldsymbol{q}} \tag{2.1}$$

The denotational semantics of a program, $[\![S]\!]$, is a so-called *superoperator* because it is a linear function mapping operators on one Hilbert space to operators on another. To be meaningful, this superoperator should map partial quantum states to partial quantum states, even when applied to a subset of the quantum variables. Formally, this means that $[\![S]\!]$ should be *completely positive and trace non-increasing*. Recall that a superoperator $\mathcal{E}$ is called *completely positive* if for every additional Hilbert space $\mathcal{H}'$ the superoperator $\mathcal{E} \otimes \mathcal{I}_{\mathcal{H}'}$ maps PSD operators to PSD operators, with $\mathcal{I}_{\mathcal{H}'}$ denoting the identity superoperator on $\mathcal{H}'$, and it is called *trace non-increasing* if $\operatorname{tr} \mathcal{E}(M) \leq \operatorname{tr} M$ for every M (it is called *trace preserving* if equality holds for all M). Conversely, these two conditions guarantee that a superoperator can be physically realized.

One can note that $[\![S]\!]$ is completely positive and trace non-increasing for any program S. Moreover, for any state $\rho \in \mathcal{D}(\mathcal{H})$, the quantity $\operatorname{tr}([\![S]\!](\rho)) \in [0, 1]$ can be interpreted as the *probability of termination* of the program S when started in the initial state ρ. In particular, the program S *terminates almost surely* (that is, with probability one) when started in state ρ if, and only if, $\operatorname{tr}([\![S]\!](\rho)) = 1$. Thus the program terminates on any input state if $[\![S]\!]$ is trace preserving (we also say that S is trace preserving). We note that (1)-(3) are always trace preserving, (4)-(6) are trace preserving if all the subprograms (S_1, S_2, S, S_ω for $\omega \in \Omega$) are trace preserving, and (7) is trace preserving if the subprogram S is trace preserving and the loop terminates with probability one.

Example 2.5 (Semantics of quantum coin toss until zero). We can use Definition 2.4 to compute the semantics for the quantum coin oss until zero program in Example 2.3. For any state $\rho \in \mathcal{D}(\mathcal{H})$,

$$\begin{aligned}
[\![A_{\text{toss-until-zero}}]\!](\rho) &= [\![\, q := |0\rangle \,; q := H(q); \textbf{while meas } q \textbf{ do } q := H(q) \textbf{ end} \,]\!](\rho) \\
&= [\![\, \textbf{while meas } q \textbf{ do } q := H(q) \textbf{ end} \,]\!]([\![\, q := H(q) \,]\!]([\![\, q := |0\rangle \,]\!](\rho))) \\
&= [\![\, \textbf{while meas } q \textbf{ do } q := H(q) \textbf{ end} \,]\!]([\![\, q := H(q) \,]\!](|0\rangle\langle 0|)) \\
&= [\![\, \textbf{while meas } q \textbf{ do } q := H(q) \textbf{ end} \,]\!](|+\rangle\langle +|).
\end{aligned}$$

using (4), (2), and (3). By (7), the semantics of the loop is for a general state $\sigma \in \mathcal{D}_{\leq 1}(\mathcal{H})$ given by

$$[\![\, \textbf{while meas } q \textbf{ do } q := H(q) \textbf{ end} \,]\!](\sigma) = \sum_{k=0}^{\infty} \Big(\mathcal{B}_0 \circ ([\![\, q := H(q) \,]\!] \circ \mathcal{B}_1)^k \Big)(\sigma),$$

where $\mathcal{B}_j(\rho) = |j\rangle\langle j|\rho|j\rangle\langle j| = \langle j|\rho|j\rangle\, |j\rangle\langle j|$. Now, for any state $\sigma \in \mathcal{D}_{\leq 1}(\mathcal{H})$, we have

$$([\![\, q := H(q) \,]\!] \circ \mathcal{B}_1)(\sigma) = \langle 1|\sigma|1\rangle\, |-\rangle\langle -|\,,$$

and therefore for any $k \geq 1$, $([\![\, q := H(q) \,]\!] \circ \mathcal{B}_1)^k(|+\rangle\langle +|) = \frac{1}{2^k}\, |-\rangle\langle -|$. Thus we find that

$$\begin{aligned}
[\![\, \textbf{while meas } q \textbf{ do } q := H(q) \textbf{ end} \,]\!](|+\rangle\langle +|) &= \mathcal{B}_0(|+\rangle\langle +|) + \sum_{k=1}^{\infty} \mathcal{B}_0\left(\frac{1}{2^k}\, |-\rangle\langle -| \right) \\
&= \frac{1}{2}\, |0\rangle\langle 0| + \sum_{k=1}^{\infty} \frac{1}{2^{k+1}}\, |0\rangle\langle 0| = |0\rangle\langle 0|\,,
\end{aligned}$$

and hence the semantics of the coin toss until zero program is, for any initial state $\rho \in \mathcal{D}(\mathcal{H})$, given by

$$[\![A_{\text{toss-until-zero}}]\!](\rho) = |0\rangle\langle 0| \tag{2.2}$$

We see that no matter what state we start in, the program always terminates in the pure state $|0\rangle$.

2.4 Quantum Hoare Logic

Hoare logic is a formal system to state and prove correctness of programs. For a program S, one specifies a precondition P and postcondition Q to form a Hoare triple $\{P\}S\{Q\}$. Such a Hoare triple is said to hold if, for any state that satisfies P, running the program on it results in a state that satisfies Q. Here P and Q are predicates over the state of the program variables. In this section, we present the formalism for *quantum Hoare logic* from [80].

Predicates. Predicates are properties of the state of the system that can hold to some degree. Recall that any PSD operator $P \preceq I$ defines a binary measurement $\{B_0, B_1\}$ by setting $B_0 = I - P$ and $B_1 = P$. We may think of P as defining a predicate: As in probabilistic Hoare logic [46], instead of assigning a definite truth value to a given predicate and state, we rather assign an *expectation* or degree to which the predicate holds in the given state – namely the probability of getting outcome 1 if one were to apply the binary measurement defined by P. For any state ρ, this probability is given by $\mathrm{tr}(P\rho)$, as explained earlier. We thus arrive at the following definition.

Definition 2.6 (Predicates and expectation). A *predicate* is a positive semidefinite operator P such that $P \preceq I$, and the set of all such predicates is denoted $\mathcal{P}_{\leq 1}(\mathcal{H})$. The *expectation* of the predicate P in a (partial) state $\rho \in \mathcal{D}_{\leq 1}(\mathcal{H})$ is defined as

$$\mathbb{E}_\rho(P) = \mathrm{tr}(P\rho) \in [0, 1].$$

We say that P *implies* Q for two predicates P, Q iff $\mathbb{E}_\rho(P) \leq \mathbb{E}_\rho(Q)$ for all states $\rho \in \mathcal{D}(\mathcal{H})$. This is equivalent to $P \preceq Q$ in the Löwner order, but is in the context of predicates often denoted $P \Rightarrow Q$.

Just like in classical Hoare logic one can also transform predicates with respect to a program S. To this end, we use the adjoint $[\![S]\!]^\dagger$ of the denotational semantics superoperator $[\![S]\!]$ (see Definition 2.4 and the discussion below it). For any superoperator $\mathcal{E}$, the adjoint $\mathcal{E}^\dagger$ satisfies the defining property that $\mathrm{tr}(A\,\mathcal{E}(B)) = \mathrm{tr}(\mathcal{E}^\dagger(A)\,B)$ for all operators A, B. We note that $\mathcal{E}$ is completely positive iff this is the case for its adjoint; it is trace preserving iff its adjoint is *unital*, that is, $\mathcal{E}^\dagger(I) = I$, and trace non-increasing if the adjoint is *sub-unital*, that is, $\mathcal{E}^\dagger(I) \preceq I$. While $[\![S]\!]$ transforms states, its adjoint $[\![S]\!]^\dagger$ naturally acts on predicates, and we have the following useful duality:

$$\mathbb{E}_{[\![S]\!](\rho)}(P) = \mathbb{E}_\rho\Big([\![S]\!]^\dagger(P)\Big).$$

for any program S and for any (partial) state $\rho \in \mathcal{D}_{\leq 1}(\mathcal{H})$ and predicate $P \in \mathcal{P}_{\leq 1}(\mathcal{H})$.

Quantum Hoare Triples. A quantum Hoare triple is denoted by

$$\{P\}S\{Q\}$$

and consists of a program S, precondition P, and postcondition Q, where P, Q are predicates as defined above. Similar to probabilistic Hoare logic, we have notions of correctness of a Hoare triple.

We start with total correctness. It states that the postcondition holds to a degree no less than the precondition:

Definition 2.7 (Total correctness). For a program S and predicates P, Q, the Hoare triple $\{P\}S\{Q\}$ is said to be *totally correct* if for all partial states $\rho \in \mathcal{D}_{\leq 1}(\mathcal{H})$,

$$\mathbb{E}_\rho(P) \leq \mathbb{E}_{[\![S]\!](\rho)}(Q).$$

We denote total correctness by $\models_{\mathrm{tot}} \{P\}S\{Q\}$. Mathematically, this is equivalent to $P \Rightarrow [\![S]\!]^\dagger(Q)$, or $P \preceq [\![S]\!]^\dagger(Q)$.

Next, we define partial correctness. Here the degree to which the postcondition holds only matters insofar as the program terminates.

Definition 2.8 (Partial correctness). For a program S and predicates P, Q, the Hoare triple $\{P\}S\{Q\}$ is said to be *partially correct* if for all partial states $\rho \in \mathcal{D}_{\leq 1}(\mathcal{H})$,

$$\mathbb{E}_\rho(P) \leq \mathbb{E}_{[\![S]\!](\rho)}(Q) + [\mathrm{tr}(\rho) - \mathrm{tr}([\![S]\!](\rho))]. \tag{2.3}$$

This condition can be equivalently stated as

$$\mathbb{E}_\rho(I - P) \geq \mathbb{E}_{[\![S]\!](\rho)}(I - Q). \tag{2.4}$$

We denote partial correctness by $\models_{\mathrm{par}} \{P\}S\{Q\}$. Mathematically, this is equivalent to $[\![S]\!]^\dagger(I-Q) \Rightarrow I - P$, or $I - P \succeq [\![S]\!]^\dagger(I - Q)$.

As explained below Definition 2.4, the term $[\mathrm{tr}(\rho) - \mathrm{tr}([\![S]\!](\rho))]$ in Eq. (2.3) is the *probability of non-termination* of the program S when started in an initial state ρ. It is always non-negative (as $[\![S]\!]$ is trace non-increasing). Intuitively, Eq. (2.3) states that the degree to which the postcondition holds is at least the degree to which the precondition holds, *minus the probability of non-termination.* The equivalent Eq. (2.4) says that the probability that the program terminates and the postcondition does *not* hold is at most the probability that the precondition does *not* hold. Total implies partial correctness.

For later use, we observe that $\models_{\mathrm{tot}} \{I\}S\{Q\}$ means that, for any initial state, the program terminates almost surely in a state satisfying the postcondition, while $\models_{\mathrm{par}} \{I\}S\{Q\}$ means that the postcondition holds whenever the program terminates. In particular, $\models_{\mathrm{tot}} \{I\}S\{I\}$ states that S terminates almost surely on any initial state, while $\models_{\mathrm{par}} \{I\}S\{I\}$ holds trivially for any program.

Example 2.9 (Hoare logic specification for quantum coin toss until zero). We now discuss a natural quantum Hoare triple for our running example (Example 2.3). One way to specify the behavior of the program $A_{\text{toss-until-zero}}$ is by the Hoare triple

$$\{I\}A_{\text{toss-until-zero}}\{|0\rangle\langle 0|\}.$$

As discussed, this states that, for any input state, the program terminates in the final state $|0\rangle$. We can verify explicitly that this Hoare triple program is totally correct. Indeed, we saw in Eq. (2.2) of Example 2.5 that $[\![A_{\text{toss-until-zero}}]\!](\rho) = |0\rangle\langle 0|$ for every state $\rho \in \mathcal{D}(\mathcal{H})$, and hence

$$\mathbb{E}_{[\![A_{\text{toss-until-zero}}]\!](\rho)}(|0\rangle\langle 0|) = \mathbb{E}_{|0\rangle\langle 0|}(|0\rangle\langle 0|) = 1 = \mathrm{tr}\,\rho = \mathbb{E}_\rho(I)$$

for any state $\rho \in \mathcal{D}(\mathcal{H})$. This confirms that the triple is totally correct. There are also much simpler programs that meet this same specification, e.g. $q := |\mathbf{0}\rangle$. See also the discussion in Appendix D.

Projections as Predicates. When the precondition in a Hoare triple is a projection, $P^2 = P$, we only need to verify correctness for pure states $\rho = |\psi\rangle\langle\psi|$ that exactly satisfy the precondition, meaning $\mathbb{E}_\rho(P) = 1$ or equivalently $P|\psi\rangle = |\psi\rangle$ [83, Theorem 3.2]. In particular, we can specify the behavior of the program when run with some initial pure state $|\Psi\rangle$ by using the precondition $P = |\Psi\rangle\langle\Psi|$: A Hoare triple $\{|\Psi\rangle\langle\Psi|\}S\{Q\}$ is totally correct iff $\mathbb{E}_{[\![S]\!](|\Psi\rangle\langle\Psi|)}(Q) = 1$, and partially correct iff $\mathbb{E}_{[\![S]\!](|\Psi\rangle\langle\Psi|)}(Q) = \mathbb{E}_{[\![S]\!](|\Psi\rangle\langle\Psi|)}(I)$. Thus:

Lemma 2.10. *Let P be a projection and Q an arbitrary predicate. Then, $P \Rightarrow Q$ holds if, and only if, $\langle\psi|Q|\psi\rangle = 1$ for every unit vector $|\psi\rangle$ such that $P|\psi\rangle = |\psi\rangle$.*

More generally, the precondition $P = |\psi\rangle\langle\psi|_{\boldsymbol{q}} \otimes I_{\boldsymbol{q}^c}$ can be used to specify the behavior on input states where the quantum variables $\boldsymbol{q}$ are in some pure state $|\psi\rangle \in \mathcal{H}_{\boldsymbol{q}}$. The situation simplifies further if the postcondition is also given by a pure state, say $Q = |\Phi\rangle\langle\Phi|$. Indeed, the Hoare triple $\{|\Psi\rangle\langle\Psi|\}S\{|\Phi\rangle\langle\Phi|\}$ is totally correct iff $[\![S]\!](|\Psi\rangle\langle\Psi|) = |\Phi\rangle\langle\Phi|$, and partially correct iff $[\![S]\!](|\Psi\rangle\langle\Psi|) = p\,|\Phi\rangle\langle\Phi|$ for some arbitrary probability of termination $p \in [0, 1]$. In other words, total (or partial) correctness of the above Hoare triple means that running the program S on state $|\Psi\rangle$

results in state $|\Phi\rangle$ (if the program terminates). We use the above observations later when specifying teleportation and search in Section 4. We caution that we *cannot* specify program behavior on mixed initial states ρ by taking $P = \rho$. Since any mixed state can be purified, this does not impose a real restriction.

2.5 Multiple Specifications

We often want to specify that a single program satisfies several Hoare triples at once. We give three motivating examples:

(1) To prove a Hoare triple $\{P\}$ **repeat** N **do** S **end** $\{Q\}$ correct, it suffices to prove that the loop body satisfies $\{R_j\}S\{R_{j+1}\}$ for all $j \in \{0, \ldots, N-1\}$, for predicates $R_0, \ldots, R_N$ with $P \Rightarrow R_0$ and $R_N \Rightarrow Q$.
(2) To assert that a program S terminates on any input, we can always add the Hoare triple $\{I\}S\{I\}$ on top of any other Hoare triple that we also want to hold (as discussed earlier).
(3) A program S that creates a qubit that, when measured, gives $x \in \{0, 1\}$ with 50% probability each, can be specified by two Hoare triples $\{I/2\}S\{|x\rangle\langle x|\}$ for $x \in \{0, 1\}$ (see Section 4.1).

At other times, we may also want to allow the program S itself to depend on some parameter. For example, the search algorithms of Section 4.3 will necessarily have to depend explicitly on the database that is being queried. We will thus consider Hoare triples

$$\{P_\lambda\}S_\lambda\{Q_\lambda\},$$

where the pre- and postconditions as well as the program are parameterized by λ in some index set Λ. Such a Hoare triple is (totally or partially) correct if it is correct for every definite value of λ. If the program S_λ does not depend on λ, we have a single program that satisfies multiple specifications, while if it depends on (part of) λ, then we have a family of programs. To stick with the literature, we will think of λ as a *formal (or meta) parameter* that is implicitly quantified over universally, but one could instead also adjust the definition of predicates to be functions $\Lambda \to \mathcal{P}_{\leq 1}(\mathcal{H})$ (and extend the notions of expectation, implication, and so forth in a straightforward way).

The following notation will be useful: if λ ranges over a finite set of options $\{\lambda_1, \ldots, \lambda_k\}$, then we will also write

$$\{P_{\lambda_1}, \ldots, P_{\lambda_k}\}S_{\lambda_1}, \ldots, S_{\lambda_k}\{Q_{\lambda_1}, \ldots, Q_{\lambda_k}\}$$

instead of $\{P_\lambda\}S_\lambda\{Q_\lambda\}$. If the precondition, the postcondition, or the program do not depend on λ, we will write the corresponding term only once. For instance, the third motivating example above could alternatively be written as $\{I/2\}S\{|0\rangle\langle 0|, |1\rangle\langle 1|\}$.

3 Correctness by Construction for Quantum Programs

In the Correctness-by-Construction (CbC) approach, one starts with a specification and successively refines it to construct a program that is guaranteed to satisfy the initial specification. Prior work on CbC defined refinement rules in terms of Hoare triples: one could replace an "abstract" program S in a Hoare triple $\{P\}S\{Q\}$ (where P, Q are some predicates) by some concrete program S' provided certain side conditions were satisfied, which often involved the validity of other Hoare triples for subprograms that have to be constructed beforehand.

Here, we extend the quantum while language by a new construct $\{P\}\square\{Q\}$, called a *hole*, which represents a yet-to-be-constructed subprogram that carries a precondition P and a postcondition Q (Section 3.1). This notion of programs with holes is similar in spirit to abstract execution [70], where one is interested in executing and analyzing programs containing "abstract statements", and also the concept of typed holes in programming [32, 51, 67]. For both partial and total correctness, we then proceed to provide *refinement rules* (Section 3.2) that can be used to construct correct

programs by replacing holes with concrete programs. Next, we prove that our refinement rules are sound: any program S constructed from a specification $\{P\}\square\{Q\}$ must satisfy that specification, meaning that the Hoare triple $\{P\}S\{Q\}$ is correct (Section 3.3). Finally, we show that our refinement rules are complete: any program S satisfying a Hoare triple $\{P\}S\{Q\}$ can be constructed from the specification $\{P\}\square\{Q\}$ (Section 3.4).

3.1 Quantum While Language with Holes

To support QbC specifications, we first extend our quantum while language in Definition 2.1 with a new construct: *holes*. A hole is a yet-to-be-constructed program tagged with a precondition and a postcondition, such that the corresponding Hoare triple should be satisfied once the hole is filled by a program. We define the syntax of the extended language below.

Definition 3.1 (Syntax). Programs in the *quantum while language with holes* are given by the following grammar:

$$\begin{aligned} S \;::=\;& \textbf{skip} \mid \boldsymbol{q} := |\mathbf{0}\rangle \mid \boldsymbol{q} := U(\boldsymbol{q}) \mid S_1; S_2 \mid \textbf{repeat } N \textbf{ do } S \textbf{ end} \\ & \mid \textbf{case meas } \boldsymbol{q} \textbf{ with } \{\omega_1 \colon M_{\omega_1}, \omega_2 \colon M_{\omega_2}, \ldots\} \textbf{ of } \omega_1 \colon S_{\omega_1},\ \omega_2 \colon S_{\omega_2},\ \ldots \textbf{ end} \\ & \mid \textbf{while meas } \boldsymbol{q} \textbf{ with } B \textbf{ do } S \textbf{ end} \mid \{P\}\square\{Q\}. \end{aligned}$$

The new construct $\{P\}\square\{Q\}$ is called a *hole* with precondition P and a postcondition Q, which are arbitrary predicates. Apart from this, the above grammar is identical to the quantum while language (Definition 2.1). A program that may contain holes is called an *abstract program*, and one that does not contain any holes is called a *concrete program*. In other words, concrete programs are simply programs in the quantum while language.

In the following we will also be interested in holes that satisfy multiple pre- and postconditions, which can be formalized just as discussed for Hoare triples (Section 2.5). We will denote these as $\{P_\lambda\}\square\{Q_\lambda\}$, where λ is some formal parameter, or use short-hand notation such as $\{P, P'\}\square\{Q, Q'\}$. See, e.g., (HP.split) and (H.repeat) below, and Remark 3.3 for further discussion.

3.2 Refinement Rules

Refinement is the process of replacing holes in abstract programs with other (abstract or concrete) programs. To this end, we define *refinement relations* on abstract programs, and we use these iteratively to construct concrete programs from specifications given by a single hole $\{P\}\square\{Q\}$. We first define refinement rules that ensure partial correctness:

Definition 3.2 (Refinement for partial correctness). We define a relation $\hookrightarrow_{\text{par}}$, called *refinement for partial correctness* on programs with holes (Definition 3.1) as follows: For any two predicates P and Q,

(H.skip) $\{P\}\square\{Q\} \hookrightarrow_{\text{par}} \textbf{skip}$, if $P \Rightarrow Q$.

(H.init) $\{P\}\square\{Q\} \hookrightarrow_{\text{par}} \boldsymbol{q} := |\mathbf{0}\rangle$, if $P \Rightarrow \sum_{\boldsymbol{x} \in \Sigma_{\boldsymbol{q}}} |\boldsymbol{x}\rangle\langle\mathbf{0}|_{\boldsymbol{q}}\, Q\, |\mathbf{0}\rangle\langle\boldsymbol{x}|_{\boldsymbol{q}}$.

(H.unit) $\{P\}\square\{Q\} \hookrightarrow_{\text{par}} \boldsymbol{q} := U(\boldsymbol{q})$, if $P \Rightarrow U_{\boldsymbol{q}}^\dagger Q U_{\boldsymbol{q}}$.

(H.seq) $\{P\}\square\{Q\} \hookrightarrow_{\text{par}} \{P\}\square\{R\}; \{R\}\square\{Q\}$ for any predicate R.

(HP.split) $\{P\}\square\{Q\} \hookrightarrow_{\text{par}} \{P_\gamma\}\square\{Q_\gamma\}$ for any two families of predicates P_γ, Q_γ for γ in some index set Γ, such that $P \Rightarrow \sum_\gamma p_\gamma P_\gamma$ and $\sum_\gamma p_\gamma Q_\gamma \Rightarrow Q$ for a probability distribution p_γ.

(H.repeat) $\{P\}\square\{Q\} \hookrightarrow_{\text{par}} \textbf{repeat } N \textbf{ do } \{R_j\}\square\{R_{j+1}\} \textbf{ end}$, where $j \in \{0, \ldots, N-1\}$, for any predicates $R_0, R_1, \ldots, R_N$ such that $P \Rightarrow R_0$ and $R_N \Rightarrow Q$.

(H.case) $\{P\}\square\{Q\} \hookrightarrow_{\text{par}} \textbf{case meas } \boldsymbol{q} \textbf{ with } \boldsymbol{M} \textbf{ of } \{\omega \colon \{P_\omega\}\square\{Q\}\}_{\omega \in \Omega} \textbf{ end}$ for any family of predicates P_ω for $\omega \in \Omega$ such that $P \Rightarrow \sum_{\omega \in \Omega} \mathcal{M}_\omega(P_\omega)$.

(HP.while) $\{P\}\square\{Q\} \hookrightarrow_{\text{par}} \textbf{while meas } \boldsymbol{q} \textbf{ with } B \textbf{ do } \{R\}\square\{\mathcal{B}_{0,\boldsymbol{q}}(Q) + \mathcal{B}_{1,\boldsymbol{q}}(R)\} \textbf{ end}$,
for any predicate R such that $P \Rightarrow \mathcal{B}_{0,\boldsymbol{q}}(Q) + \mathcal{B}_{1,\boldsymbol{q}}(R)$.

We also have rules for composite statements:

(C.seqL) $S_1' ; S_2 \hookrightarrow_{\text{par}} S_1 ; S_2$, if $S_1' \hookrightarrow_{\text{par}} S_1$.

(C.seqR) $S_1 ; S_2' \hookrightarrow_{\text{par}} S_1 ; S_2$, if $S_2' \hookrightarrow_{\text{par}} S_2$.

(C.repeat) $\textbf{repeat } N \textbf{ do } S' \textbf{ end} \hookrightarrow_{\text{par}} \textbf{repeat } N \textbf{ do } S \textbf{ end}$, if $S' \hookrightarrow_{\text{par}} S$.

(C.case) $\textbf{case meas } \boldsymbol{q} \textbf{ with } \boldsymbol{M} \textbf{ of } \omega\colon S_\omega' \ , \ \ldots \textbf{ end} \hookrightarrow_{\text{par}}$
$\textbf{case meas } \boldsymbol{q} \textbf{ with } \boldsymbol{M} \textbf{ of } \omega\colon S_\omega \ , \ \ldots \textbf{ end}$,
if $S_\omega' \hookrightarrow_{\text{par}} S_\omega$ for one $\omega \in \Omega$ (and the rest unchanged).

(C.while) $\textbf{while meas } \boldsymbol{q} \textbf{ with } B \textbf{ do } S' \textbf{ end} \hookrightarrow_{\text{par}} \textbf{while meas } \boldsymbol{q} \textbf{ with } B \textbf{ do } S \textbf{ end}$,
if $S' \hookrightarrow_{\text{par}} S$.

For any two programs S, S', we say S' *refines in one step to* S *ensuring partial correctness* if $S' \hookrightarrow_{\text{par}} S$. More generally, for any $k \geq 0$ we define $S' \hookrightarrow_{\text{par}}^{k} S$ if S' refines to S in k such steps. We say S' *refines to* S *ensuring partial correctness* if S can be obtained from S' by applying any number of refinement steps and denote this by $S' \hookrightarrow_{\text{par}}^{*} S$. Clearly, $\hookrightarrow_{\text{par}}^{*} = \bigcup_{k=0}^{\infty} \hookrightarrow_{\text{par}}^{k}$ is the reflexive and transitive closure of the relation $\hookrightarrow_{\text{par}}$.

In Definition 3.2, the rules labeled (H.∗) and (HP.∗) are used to refine a single hole to another program (H stands for hole). The rules labeled (C.∗) are used to refine holes in composite programs (C stands for composite). See Section 4.1 and Appendix D for pedagogical expositions on applying refinements to construct quantum coin-tossing programs. The first three rules refine to concrete program statements: (H.skip) refines to a **skip** statement, (H.init) refines to an initialization $\boldsymbol{q} := |\boldsymbol{0}\rangle$, and (H.unit) refines to a unitary application $\boldsymbol{q} := U(\boldsymbol{q})$. To refine a hole to a sequence of two holes, we can use the (H.seq) rule with any arbitrary intermediate condition R. To motivate (HP.split), observe that if a program satisfies two Hoare triples $\{P\}S\{Q\}$ and $\{P'\}S\{Q'\}$, then it also satisfies any combination $\{pP + (1-p)Q\}S\{pQ + (1-p)Q'\}$ for $p \in [0,1]$. So to ensure the latter it suffices to construct a program that satisfies the former. Note that in stating this rule we use the syntax for multiple specifications discussed below Definition 3.1. In (HP.split) we also allow for weakening preconditions and strengthening postconditions; we isolate this in (H.sw) below for convenience. To refine a hole to a **repeat** statement using (H.repeat), we must find a family of predicates $R_0, R_1, \ldots, R_N$, such that R_n holds after running the loop body $0 \leq n \leq N$ times, with R_0 implied by the precondition and R_N implying the postcondition. The body of the **repeat** statement is $\{R_j\}\square\{R_{j+1}\}$ where $j \in \{0, \ldots, N-1\}$ is a formal parameter, meaning that the yet-to-be-constructed program must satisfy these specification for all such j. To refine a hole to a **case** statement using (H.case), we must find a family of predicates P_ω for each measurement outcome ω such that if P holds before the measurement then P_ω holds after measurement upon seeing outcome ω. Finally, to refine a hole to a **while** statement using (HP.while), we need to find an "invariant" R for the loop body. If the measurement succeeds after the execution of the loop body, then R must hold; otherwise Q must hold. Finally, the composite rules allow refining any hole in a composite program using any of the rules above.

Remark 3.3 (Multiple specifications and formal parameters). As mentioned, some refinement rules can introduce holes with multiple specifications. For example, (H.repeat) introduces a hole with pre- and postcondition labeled by an index $j \in \{0, \ldots, N-1\}$, and (HP.split) introduces a new parameter γ in some arbitrary index set Γ. We can model this formally by implicitly extending the index set Λ of Section 2.5 to include this new parameter. Later refinements may depend on this

parameter. For example, if we refine a hole produced by (H.repeat) with the (H.seq) rule, then the intermediate predicates R may also depend on j. See Section 4 for many examples.

We can also apply these rules to the syntactic sugar introduced in Section 2.3 and deduce some other rules for convenience. For example, since an **if** statement is shorthand for a **case** statement, we also have

(H.ifElse) $\{P\}\square\{Q\} \hookrightarrow_{\text{par}}$ **if meas** $\boldsymbol{q}$ **with** B **then** $\{R_1\}\square\{Q\}$ **else** $\{R_0\}\square\{Q\}$ **end**,
for any predicates R_0, R_1 s.th. $P \Rightarrow \mathcal{B}_{0,\boldsymbol{q}}(R_0) + \mathcal{B}_{1,\boldsymbol{q}}(R_1)$.

We can also introducing an **if** statement without an **else** branch:

(H.if) $\{P\}\square\{Q\} \hookrightarrow^*_{\text{par}}$ **if meas** $\boldsymbol{q}$ **with** B **then** $\{R\}\square\{Q\}$ **end**,
for any predicate R such that $P \Rightarrow \mathcal{B}_{0,\boldsymbol{q}}(Q) + \mathcal{B}_{1,\boldsymbol{q}}(R)$.

This follows from (H.ifElse) by further refining the second hole using (H.skip).

Finally, we have the following rule that shows that we may always construct a program that has a weaker precondition and a stronger postcondition:

(H.sw) $\{P\}\square\{Q\} \hookrightarrow_{\text{par}} \{P'\}\square\{Q'\}$ for any predicates P', Q' such that $P \Rightarrow P'$ and $Q' \Rightarrow Q$.

This is a special case of (HP.split) where we take the two families to consist of a single predicate P' and Q', respectively.

Remark 3.4 (Challenges of quantum verification). The quantum setting poses some interesting new challenges. For example, in a (H.case) rule for classical programs, one can always choose the predicates as $P_\omega = P \wedge (q = \omega)$. In the quantum setting, there is generally no canonical choice of the predicates P_ω. We leave the problem of finding suitable heuristics to future work.

We also define refinement rules that ensure total correctness:

Definition 3.5 (Refinement for total correctness). We define a relation $\hookrightarrow_{\text{tot}}$, called *refinement for total correctness* on programs with holes, by using all the rules from Definition 3.2 that are labeled (H.∗) or (C.∗), by replacing each $\hookrightarrow_{\text{par}}$ with $\hookrightarrow_{\text{tot}}$, but replacing (HP.while) and (HP.split) by the following rules respectively:

(HT.while) $\{P\}\square\{Q\} \hookrightarrow_{\text{tot}}$ **while meas** $\boldsymbol{q}$ **with** B **do** $\{R_{n+1}\}\square\{\mathcal{B}_{0,\boldsymbol{q}}(Q) + \mathcal{B}_{1,\boldsymbol{q}}(R_n)\}$ **end**,
for any binary measurement B and sequence of predicates $\{R_n\}_{n\in\mathbb{N}}$ that is weakly increasing in the sense that $R_n \Rightarrow R_{n+1}$ for all $n \in \mathbb{N}$, such that $R_0 = 0$ and the limit $R := \lim_{n\to\infty} R_n$ satisfies $P \Rightarrow \mathcal{B}_{0,\boldsymbol{q}}(Q) + \mathcal{B}_{1,\boldsymbol{q}}(R)$.

(HT.split) $\{P\}\square\{Q\} \hookrightarrow_{\text{tot}} \{P_\gamma\}\square\{Q_\gamma\}$ for any two families of predicates P_γ, Q_γ for γ in some index set Γ, such that $P \Rightarrow \sum_\gamma p_\gamma P_\gamma$ and $\sum_\gamma p_\gamma Q_\gamma \Rightarrow Q$ for some $p_\gamma \geq 0$.

For any two programs S, S', we say that S' *refines in one step to* S *ensuring total correctness* if $S' \hookrightarrow_{\text{tot}} S$. More generally, for any $k \geq 0$, we define $S' \hookrightarrow^k_{\text{tot}} S$ if S' refines to S in k such steps. We say that S' *refines to* S *ensuring total correctness* if S can be obtained from S' by applying any number of refinement steps and denote this by $S' \hookrightarrow^*_{\text{tot}} S$. Similarly as above, $\hookrightarrow^*_{\text{tot}} = \bigcup_{k=0}^{\infty} \hookrightarrow^k_{\text{tot}}$ is the reflexive and transitive closure of the relation $\hookrightarrow_{\text{tot}}$.

The new (HT.while) rule can be used to construct **while** loops that are totally correct. To understand it intuitively, note that the subprogram obtained by unrolling the loop body n times (ignoring the initial measurement) satisfies the specification $\{R_n\}\square\{\mathcal{B}_0(Q)\}$, This can be interpreted as follows: if we start inside the loop and R_n holds (with some probability), then the loop terminates within n iterations in a state that satisfies Q (with at least that probability). We remark that the limit R (which is easily seen to always exist) precisely satisfies the requirements on the "loop invariant" of the (HP.while) rule. See Section 4 for examples and Section 1.2 for related work on proving termination of probabilistic programs using loop invariant predicates.

The (HT.split) rule is more general than (HP.split) as it allows arbitrary non-negative p_ω that need not add up to one.

3.3 Soundness of Refinement

We now show that the sets of rules given above are *sound*, meaning that each ensures the correctness of constructed programs, in the following sense: if one starts with a specification, that is, a single hole $\{P\}\square\{Q\}$, and repeatedly refines to construct a concrete program S (i.e., a program without holes), then the constructed program satisfies the initial specification, meaning that Hoare triple $\{P\}S\{Q\}$ is correct. This holds for both partial and total correctness:

Theorem 3.6 (Soundness of refinement for partial correctness). *For any two predicates P, Q and any concrete program S, if $\{P\}\square\{Q\} \hookrightarrow^*_{\text{par}} S$, then $\models_{par} \{P\}S\{Q\}$.*

Theorem 3.7 (Soundness of refinement for total correctness). *For any two predicates P, Q and any concrete program S, if $\{P\}\square\{Q\} \hookrightarrow^*_{\text{tot}} S$ then $\models_{tot} \{P\}S\{Q\}$.*

Theorems 3.6 and 3.7 are proved by induction over the length of the refinement chain, with the (HP.while) and (HT.while) rules being most delicate. The detailed proofs can be found in Appendix A.

3.4 Completeness of Refinement

We now show that the refinement rules given above are *complete*, meaning that they allow us to construct any correct program, in the following sense: if one has a concrete program S (i.e., a program without holes) such that the Hoare triple $\{P\}S\{Q\}$ is correct, then the program can be constructed from $\{P\}\square\{Q\}$ by applying a finite number of refinements. Formally, we have the following results:

Theorem 3.8 (Completeness of refinement for partial correctness). *For any two predicates P, Q and any concrete program S, if $\models_{par} \{P\}S\{Q\}$ then $\{P\}\square\{Q\} \hookrightarrow^*_{\text{par}} S$.*

Theorem 3.9 (Completeness of refinement for total correctness). *For any two predicates P, Q and any concrete program S, if $\models_{tot} \{P\}S\{Q\}$ then $\{P\}\square\{Q\} \hookrightarrow^*_{\text{tot}} S$.*

Theorems 3.8 and 3.9 can be proved by induction on the structure of the program, with the **while** construct requiring a careful analysis. The detailed proofs can be found in Appendix B.

4 Examples

In this section, we demonstrate how to use the QbC approach to construct quantum programs from their specification. We first discuss a pedagogical example of a fair quantum coin. We then construct a quantum teleportation protocol (Section 4.2) and two quantum search algorithms (Section 4.3). We then describe new refinement rules to boost the success probability of quantum algorithms (Section 4.4). This illustrates how QbC can be usefully extended by higher-level algorithmic patterns and construction principles. Particularly for the more complicated algorithms, we find that the QbC approach allows naturally discovering program detail on the fly, without explicitly using a priori knowledge of the final algorithms. It also suggests key design decisions that give rise to different quantum programs satisfying the same specification. Additionally, in Appendix D we discuss how the running example from Section 2 can be constructed from a structured specification.

4.1 Fair Quantum Coin

A fair quantum coin is a program that prepares a quantum bit in a state that, when measured, gives rise to either outcome with 50% probability. There are infinitely many such states, but two natural ones are the *Hadamard basis* states $|\pm\rangle = (|0\rangle \pm |1\rangle)/\sqrt{2}$. We use a single qubit quantum variable q.

Specification. To specify that each outcome occurs with 50% probability, we can use the following program with a single hole:

$$S_{\text{fair-coin}} = \{I/2\}\square\{|x\rangle\langle x|\}\,,$$

where $x \in \{0,1\}$ is a formal parameter (see Section 2.5). Indeed, suppose we manage to refine the above into a program S' without holes which does not explicitly use the parameter x. Then our soundness result (Theorem 3.7) guarantees that the Hoare triple

$$\{I/2\}S'\{|x\rangle\langle x|\}$$

is valid for every $x \in \{0,1\}$, meaning that if we run the program (on an arbitrary state) and measure the qubit, we obtain either outcome $x \in \{0,1\}$ with probability at least, and hence equal to $\frac{1}{2}$.

Construction. The well-known idea is that the Hadamard gate

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{4.1}$$

maps the standard basis to the Hadamard basis, which allows us to realize the coin toss. To confirm this, we consider the following sequence of refinements:

$$\begin{aligned} S_{\text{fair-coin}} &\hookrightarrow_{\text{tot}} \{I/2\}\square\{R\}\,;\{R\}\square\{|x\rangle\langle x|\} && \text{(H.seq)}\\ &\hookrightarrow_{\text{tot}} q := |0\rangle\,;\{R\}\square\{|x\rangle\langle x|\} && \text{(H.init)}\\ &\hookrightarrow_{\text{tot}} q := |0\rangle\,;q := H(q) && \text{(H.unit)} \end{aligned}$$

The first refinement is always valid, but how should we pick the predicate R so that the subsequent refinements can be applied? To apply (H.init), we need that $I/2 \Rightarrow \sum_{y\in\{0,1\}} |y\rangle\langle 0|\, R\, |0\rangle\langle y|$, meaning that $\langle 0|R|0\rangle \geq \frac{1}{2}$ for $x \in \{0,1\}$. To apply (H.unit), we should choose R such that $R \Rightarrow H\,|x\rangle\langle x|\,H$. Since the latter is a pure state, this suggests $R = H\,|x\rangle\langle x|\,H$ (which is also the weakest precondition for the Hadamard subprogram and postcondition), and for this choice we have that $\langle 0|R|0\rangle = |\langle 0|H|x\rangle|^2 = \frac{1}{2}$. Thus the above refinements are valid and we have constructed a program that implements the fair coin toss specification correctly, by construction.

$$S_{\text{fair-coin}} \hookrightarrow^*_{\text{tot}} q := |0\rangle\,;q := H(q)\,,$$

4.2 Quantum Teleportation

Imagine two parties, Alice and Bob, who share a maximally entangled state, say qubits a, b in state

$$|\phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle). \tag{4.2}$$

Alice has another qubit q in an unknown *quantum* state, and wants to transfer its state to Bob's qubit b, by sending only *classical* information but utilizing the maximally entangled state as a resource. Furthermore, if q was correlated or entangled with other quantum variables, then after teleportation the same should be true for b. This is called *quantum teleportation* and is a basic building block for quantum communication (see, e.g., Nielsen and Chuang [48]).

Specification. To specify teleportation, consider an arbitrary quantum state ρ between Alice's and some arbitrary other quantum variable r, which does not participate in the protocol (in quantum information, r is called a reference system). After teleportation, we want the state of Bob's qubit b and r to be in the same state. Without loss of generality, we can take ρ to be a maximally entangled state, $\rho = |\phi^+\rangle\langle\phi^+|$ (that is, it suffices to realize "entanglement swapping"). Indeed, a basic principle of quantum information theory asserts that if two quantum programs (completely positive maps) have the same behavior when applied to one half of a maximally entangled state, then they must

have the same behavior on all states [48, 76]. Recall from Lemma 2.10 that we can assert that quantum variables are in a given pure state by using its projection as the predicate. This translates to the following initial specification:

$$S_{\text{teleport}} = \{|\phi^+\rangle\langle\phi^+|_{qr} \otimes |\phi^+\rangle\langle\phi^+|_{ab}\} \square \{|\phi^+\rangle\langle\phi^+|_{br}\} \tag{4.3}$$

Now, an arbitrary program that meets the above specification will not be a teleportation protocol, since in teleportation we want to constrain all quantum operations to Alice and Bob's variables, while only allowing them to communicate classical bits from Alice to Bob. This disallows, e.g., simply swapping qubits q and b by applying a quantum gate. We can implement this latter constraint by the following sequence of refinements to obtain a program with two holes, where the first hole will be further refined by a program acting only on Alice's qubits q and a, and the second hole by a program acting only on Bob's qubit b:

$$S_{\text{teleport}} \hookrightarrow_{\text{tot}} \{|\phi^+\rangle\langle\phi^+|_{qr} \otimes |\phi^+\rangle\langle\phi^+|_{ab}\} \square \{P\}; \{P\} \square \{|\phi^+\rangle\langle\phi^+|_{br}\} \tag{H.seq}$$

$$\hookrightarrow_{\text{tot}} \begin{array}{l} \{|\phi^+\rangle\langle\phi^+|_{qr} \otimes |\phi^+\rangle\langle\phi^+|_{ab}\} \square \{P\}; \\ \textbf{case meas } q, a \textbf{ of } (x, y) \colon \{Q(x, y)\} \square \{|\phi^+\rangle\langle\phi^+|_{br}\} \textbf{ end} \end{array}, \tag{H.case}$$

where $x, y \in \{0, 1\}$ for arbitrary predicates $Q(x, y)$. We can ensure that (H.case) is valid by taking

$$P := \sum_{x,y} |xy\rangle\langle xy|_{qa}\, Q(x, y)\, |xy\rangle\langle xy|_{qa}\,. \tag{4.4}$$

We have arrived at the specification S'_{teleport} which refines Eq. (4.3),

$$S'_{\text{teleport}} = \begin{array}{l} \underbrace{\{|\phi^+\rangle\langle\phi^+|_{qr} \otimes |\phi^+\rangle\langle\phi^+|_{ab}\} \square \{P\};}_{S_{\text{Alice}}} \\ \textbf{case meas } q, a \textbf{ of } (x, y) \colon \underbrace{\{Q(x, y)\} \square \{|\phi^+\rangle\langle\phi^+|_{br}\}}_{S_{\text{Bob},x,y}} \textbf{ end} \end{array}$$

which consists of three steps:

(1) Alice first applies some quantum program S_{Alice} (which will be constructed to only act on her qubits).
(2) Alice measures her qubits q, a in the standard basis (without loss of generality) and obtains as outcomes two classical bits x, y, which we imagine she communicates to Bob.
(3) Bob applies another quantum program $S_{\text{Bob},x,y}$ (which will be constructed to only act on his qubit) that is allowed to explicitly depend on the outcomes x and y.

This precisely captures the structure as well as functionality of a quantum teleportation protocol.

Construction. Before starting the construction we first simplify the precondition $Q(x, y)$ of Bob's program. As the latter is run straight after Alice's measurements, which yielded outcomes x, y, we know that Alice's qubits must be in state $|x, y\rangle$. This motivates and in fact implies that we may take

$$Q(x, y) = |xy\rangle\langle xy|_{qa} \otimes R_{br}(x, y) \tag{4.5}$$

for certain predicates $R_{br}(x, y)$ on qubits b, r that still need to be determined.

To construct Alice and Bob's programs, we make the straightforward guess that each applies some unitary, which in Bob's case may depend on the measurement outcomes x, y. Thus we refine

$$S_{\text{Alice}} \hookrightarrow_{\text{tot}} q, a := V(q, a) \qquad \text{and} \qquad S_{\text{Bob},x,y} \hookrightarrow_{\text{tot}} b := U(x, y)(b)\,, \tag{H.unit}$$

where V is a two-qubit unitary and the $U(x,y)$ are one-qubit unitaries that we still need to construct. The second refinement is valid assuming $U_b(x,y)Q(x,y)U_b^\dagger(x,y) \Rrightarrow |\phi^+\rangle\langle\phi^+|_{br}$, which by Eq. (4.5) we can satisfy by picking

$$R_{br}(x,y) = U_b^\dagger(x,y)\,|\phi^+\rangle\langle\phi^+|_{br}\,U_b(x,y).$$

By Eq. (4.4), this in turn implies that

$$P = \sum_{x,y} |xy\rangle\langle xy|_{qa} \otimes U_b^\dagger(x,y)\,|\phi^+\rangle\langle\phi^+|_{br}\,U_b(x,y). \tag{4.6}$$

The first refinement is valid if $V_{qa}(|\phi^+\rangle\langle\phi^+|_{qr} \otimes |\phi^+\rangle\langle\phi^+|_{ab})V_{qa}^\dagger \Rrightarrow P$. As the left-hand side is a pure state, Lemma 2.10 shows that this condition is equivalent to

$$\begin{aligned} 1 = \langle\phi^+_{qr}\otimes\phi^+_{ab}|V_{qa}^\dagger P V_{qa}|\phi^+_{qr}\otimes\phi^+_{ab}\rangle &= \sum_{x,y} |\langle x_q \otimes y_a \otimes \phi^+_{br}|U_b(x,y)V_{qa}|\phi^+_{qr}\otimes\phi^+_{ab}\rangle|^2 \\ &= \sum_{x,y} |\langle x_q \otimes y_a \otimes \phi^+_{br}|U_b(x,y)V_{rb}^T|\phi^+_{qr}\otimes\phi^+_{ab}\rangle|^2 \\ &= \frac{1}{4}\sum_{x,y} |\langle\phi^+_{br}|U_b(x,y)V_{rb}^T|x_r\otimes y_b\rangle|^2, \end{aligned} \tag{4.7}$$

where we first used Eq. (4.6) and then the identity $V_{qa}\,|\phi^+_{qr}\otimes\phi^+_{ab}\rangle = V_{rb}^T\,|\phi^+_{qr}\otimes\phi^+_{ab}\rangle$, known as the "transpose trick", which allows moving an arbitrary operator acting on qubits q, a to the other side of the maximally entangled states, that is, to act on qubits r, b, if we replace the operator by its transpose (in the computational basis). This well-known identity is easily verified by direct calculation. The final step follows by observing that $\langle x_q|\,|\phi^+_{qr}\rangle = |x_r\rangle/\sqrt{2}$ and similarly $\langle y_a|\,|\phi^+_{ab}\rangle = |x_b\rangle/\sqrt{2}$. Since all of the summands in Eq. (4.7) are at most one, they must all be equal to one. In other words,

$$V_{rb}^T\,|xy\rangle_{rb} \quad \text{and} \quad U_b^\dagger(x,y)\,|\phi^+_{br}\rangle$$

must be the same states for all $x, y \in \{0,1\}$ (up to irrelevant overall phases). Note that the left-hand side states make up an orthonormal basis, while the right-hand states are all maximally entangled (since they obtained by applying a unitary to one of the qubits of $|\phi^+\rangle$). It follows that we should pick V_{rb}^T to be a unitary that maps the standard basis to a basis of maximally entangled states. And this is also sufficient since any two maximally entangled states differ by a unitary on either of the qubits. As is well known, the Bell basis consists of maximally entangled states and it can be prepared by the unitary $V_{rb}^T = \mathrm{CNOT}_{rb}H_b$. Thus we take

$$V = (H\otimes I)\mathrm{CNOT}.$$

Finally, we note that the Bell states can be obtained from the standard maximally entangled state as $V_{rb}^T\,|yx\rangle_{rb} = Z_b^y X_b^x\,|\phi^+\rangle_{rb}$, where X and Z denote the Pauli X and Z matrices. Thus Bob's unitaries should be

$$U(x,y) = X^x Z^y.$$

Altogether, we have constructed the following program, which is nothing but the standard protocol for quantum teleportation:

$S'_{\text{teleport}} \hookrightarrow^*_{\text{tot}}$
$q, a := (H\otimes I)\mathrm{CNOT}(q,a);$
case meas (q,a) **of** $00\colon b := I(b); 01\colon b := Z(b); 10\colon b := X(b); 11\colon b := XZ(b)$ **end**

As it was obtained by refinement, it satisfies the specification by construction.

4.3 Quantum Search

We consider the following search problem [30, 48]. Given query access to a Boolean function or "database" $f\colon \{0,1\}^n \to \{0,1\}$, we wish to find a bitstring $x \in \{0,1\}^n$ such that $f(x) = 1$. Such an x is often called a "solution" or "marked element". In the quantum setting, we are given query access to f via the following *standard quantum oracle* unitary, $O_f \left|x\right\rangle_{\boldsymbol{q}} \left|y\right\rangle_a = \left|x\right\rangle_{\boldsymbol{q}} \left|y \oplus f(x)\right\rangle_a$, or by the following *phase oracle* unitary

$$P_f \left|x\right\rangle_{\boldsymbol{q}} = (-1)^{f(x)} \left|x\right\rangle_{\boldsymbol{q}}, \tag{4.8}$$

which can be obtained from the former in a straightforward fashion. Here, $\boldsymbol{q}$ is a quantum variable consisting of n qubits. Let us define $N = 2^n$ as the size of the search space, and $T = |\{x : f(x) = 1\}|$ as the number of solutions. In the example constructions below, we will assume knowledge of this number of solutions. In the following, we will first present a specification of the search problem, and then construct *two* different programs that satisfy the specification by construction: a simple algorithm based on random sampling and Grover's celebrated quantum search algorithm [30].

Specification. The search problem can be specified as follows:

$$S_{\text{search}}(p) = \{pI\} \square \{\textstyle\sum_{x:f(x)=1} \left|x\right\rangle\!\left\langle x\right|_{\boldsymbol{q}}\},$$

It states that if we measure the state after program execution, we obtain a solution x with probability at least p. Thus, p is the probability of success of the search algorithm. In our constructions below, we treat p as a parameter that will naturally be selected during the refinement process. Note that whatever the value of p, such an algorithm (if it terminates) can always be amplified or "boosted" to any desired success probability by repeating it sufficiently often until it finds a solution. We discuss this in Section 4.4 below and propose refinement rules to automate this reasoning.

Construction I: Random Sampling. One can solve the search problem by sampling $x \in \{0,1\}^n$ uniformly at random. Clearly, this succeeds with probability T/N. One way to achieve this by a quantum program is by preparing the uniform superposition $\left|U\right\rangle = \frac{1}{\sqrt{N}} \sum_{x\in\{0,1\}^n} \left|x\right\rangle$, since measuring this state in the standard basis will yield a uniformly random $x \in \{0,1\}^n$. We can confirm that this construction works with the aforementioned success probability, by refining the specification into a program that prepares the state $\left|U\right\rangle$. This can be done by first initializing all qubits in the zero state and then applying Hadamard gates H (Eq. (4.1)) to all of the qubits:

$$\begin{aligned} S_{\text{search}}(p) &\hookrightarrow_{\text{tot}} \{pI\} \square \{R\}; \{R\} \square \{\textstyle\sum_{x:f(x)=1} \left|x\right\rangle\!\left\langle x\right|_{\boldsymbol{q}}\} && \text{(H.seq)} \\ &\hookrightarrow_{\text{tot}} \boldsymbol{q} := \left|\boldsymbol{0}\right\rangle; \{R\} \square \{\textstyle\sum_{x:f(x)=1} \left|x\right\rangle\!\left\langle x\right|_{\boldsymbol{q}}\} && \text{(H.init)} \\ &\hookrightarrow_{\text{tot}} \boldsymbol{q} := \left|\boldsymbol{0}\right\rangle; \boldsymbol{q} := H^{\otimes n}(\boldsymbol{q}) && \text{(H.unit)} \end{aligned}$$

To apply (H.init), we need that $pI \Rightarrow \sum_x \left|x\right\rangle\!\left\langle 0\right|_{\boldsymbol{q}} R \left|0\right\rangle\!\left\langle x\right|_{\boldsymbol{q}}$, that is, $p \le \langle 0|R|0\rangle$. To apply (H.unit), we can choose

$$R = H^{\otimes n} \left(\sum_{x:f(x)=1} \left|x\right\rangle\!\left\langle x\right|_{\boldsymbol{q}} \right) H^{\otimes n}.$$

Thus the above refinements are valid if

$$p \le \langle 0|R|0\rangle = \langle U| \sum_{x:f(x)=1} \left|x\right\rangle\!\left\langle x\right|_{\boldsymbol{q}} \left|U\right\rangle = \frac{T}{N}.$$

Thus we should pick $p := T/N$ to maximize the success probability. Altogether, we have constructed a search algorithm that succeeds with probability $p = T/N$. As mentioned earlier and will be discussed in detail in Section 4.4, by repeating the above $O(N/T)$ times until we find a solution,

we can obtain a program that solves the search problem with any desired constant probability of success (say, $p = 2/3$) at a cost of $O(N/T)$ queries.

Construction II: Grover Search. Grover [30] proposed a quantum algorithm for the search problem which gives a quadratic speedup over random sampling. In the following, we will re-construct this algorithm by making natural choices using one key idea at a time.

Step 1. The first observation is that while the uniform superposition $|U\rangle$ over all bitstrings is easy to prepare, what we are really after is the uniform superposition of all *solutions*, that is, the "good" state $|G\rangle = \frac{1}{\sqrt{T}} \sum_{x \text{ s.t. } f(x)=1} |x\rangle$. Indeed, measuring $|G\rangle$ will yield a solution with probability one. We can formalize this idea by strengthening the postcondition of the specification:

$$S_{\text{search}}(p) \hookrightarrow_{\text{tot}} \{pI\} \square \{|G\rangle\langle G|\} =: S_{\text{good}}(p), \tag{H.sw}$$

We may apply (H.sw) since the condition $|G\rangle\langle G| \Rightarrow \sum_{x \text{ s.t. } f(x)=1} |x\rangle\langle x|$ is satisfied. Indeed, the good state $|G\rangle$ is clearly contained in the span of the basis states $|x\rangle$ corresponding to solutions x.

Step 2. The uniform state $|U\rangle$ (which is easy to prepare, but not very useful) and the good state $|G\rangle$ (which solves the problem, but is a priori unclear how to prepare) span a two-dimensional subspace of the exponentially large Hilbert space. The key idea (which has no classical counterpart) then is to try to *rotate* the state $|U\rangle$ onto $|G\rangle$ in this two-dimensional subspace.

To realize this idea, we first define an orthonormal basis of the two-dimensional subspace by picking $|G\rangle$ and a vector orthogonal to it, namely the "bad state" $|B\rangle = \frac{1}{\sqrt{N-T}} \sum_{x \text{ s.t. } f(x)=0} |x\rangle$. At any stage of the program, we would like the state of $\boldsymbol{q}$ to be of the form

$$|\theta\rangle = \cos\theta\,|B\rangle + \sin\theta\,|G\rangle$$

for some angle θ, which will serve as a loop variant in the following. The plan is now to prepare the uniform state, which has angle $v = \arcsin\sqrt{T/N} \in [0, \pi/2]$ as it can be written as $|U\rangle = \sqrt{(N-T)/N}\,|B\rangle + \sqrt{T/N}\,|G\rangle$, and then rotate it repeatedly by some angle $\delta > 0$ towards the good state $|G\rangle$, which is at angle $\frac{\pi}{2}$. We can formalize this by the following refinements:

$$\begin{aligned} S_{\text{good}}(p) &\hookrightarrow_{\text{tot}} \{pI\} \square \{|\theta_0\rangle\langle\theta_0|\}; \{|\theta_0\rangle\langle\theta_0|\} \square \{|\tfrac{\pi}{2}\rangle\langle\tfrac{\pi}{2}|\} && \text{(H.seq)} \\ &\hookrightarrow_{\text{tot}} \underbrace{\{pI\} \square \{|\theta_0\rangle\langle\theta_0|\}}_{S_{\text{init}}}; \textbf{repeat } r \textbf{ do } \underbrace{\{|\theta_j\rangle\langle\theta_j|\} \square \{|\theta_{j+1}\rangle\langle\theta_{j+1}|\}}_{S_{\text{rotate}}(\delta)} \textbf{ end} && \text{(H.repeat)} \end{aligned}$$

To apply (H.repeat), we choose $\theta_r = \frac{\pi}{2}$ and $\theta_{j+1} = \theta_j + \delta$ for some arbitary rotation angle δ and number of rotations r that we will determine later. Hence $\theta_0 = \frac{\pi}{2} - r\delta$.

We first construct the S_{init} program. Following the plan, we prepare the uniform superposition $|U\rangle = |v\rangle$, which we already know how to do from above:

$$\begin{aligned} S_{\text{init}} &\hookrightarrow_{\text{tot}} \{pI\} \square \{R\}; \{R\} \square \{|\theta_0\rangle\langle\theta_0|\} && \text{(H.seq)} \\ &\hookrightarrow_{\text{tot}} \boldsymbol{q} := |\mathbf{0}\rangle\,; \{R\} \square \{|\theta_0\rangle\langle\theta_0|\} && \text{(H.init)} \\ &\hookrightarrow_{\text{tot}} \boldsymbol{q} := |\mathbf{0}\rangle\,; \boldsymbol{q} := H^{\otimes n}(\boldsymbol{q}) && \text{(H.unit)} \end{aligned}$$

Since the postcondition is different from the above, we also need to choose R differently, but we can follow the same reasoning. In order to apply (H.init), we need that $p \le \langle 0|R|0\rangle$, and to apply (H.unit) we can choose $R = H^{\otimes n}\,|\theta_0\rangle\langle\theta_0|\,H^{\otimes n}$. Together, we find that the maximum success probability for which the above refinements are valid is given by

$$p := \langle 0|R|0\rangle = |\langle 0|H^{\otimes n}|\theta_0\rangle|^2 = |\langle v|\theta_0\rangle|^2 = \cos^2\Big(\frac{\pi}{2} - r\delta - v\Big).$$

To maximize this probability, we should further choose r such that the right-hand side is maximized. We will pick r such that the angle in the cosine is closest to 0:

$$r := \left[\frac{\frac{\pi}{2} - v}{\delta}\right], \tag{4.9}$$

where $[\cdot]$ rounds to the nearest integer. Clearly, $p \geq \cos^2(\delta/2)$.

Step 3. We still need to construct the program $S_{\text{rotate}}(\delta)$ for some rotation angle δ. To this end, we first observe that it suffices to construct a program that satisfies the stronger specification:

$$S'_{\text{rotate}}(\delta) = \{|\theta\rangle\langle\theta|\} \square \{|\theta+\delta\rangle\langle\theta+\delta|\}.$$

Indeed, programs satisfying this specification rotate *all* states $|\theta\rangle$ in the two-dimensional subspace by δ, as opposed just the states $|\theta_j\rangle$ for $j \in \{0, 1, \ldots, N-1\}$.

How can we obtain such a rotation? Observe that the quantum phase oracle P_f in Eq. (4.8) is a *reflection* about the vector $|B\rangle$, as it maps $P_f|\theta\rangle = |-\theta\rangle$. Now, we know that two reflections make a rotation. For our second reflection we simply pick some known state (independent of the instance of the search problem) to reflect about. A natural choice is $|U\rangle$, since the corresponding reflection $2|U\rangle\langle U| - I$ can be efficiently implemented using $O(n)$ gates. Thus we introduce these two reflections in sequence and determine the rotation angle from the conditions of the refinements:

$$\begin{aligned}
S'_{\text{rotate}}(\delta) &\hookrightarrow_{\text{tot}} \{|\theta\rangle\langle\theta|\} \square \{|-\theta\rangle\langle-\theta|\}; \{|-\theta\rangle\langle-\theta|\} \square \{|\theta+\delta\rangle\langle\theta+\delta|\} && \text{(H.seq)}\\
&\hookrightarrow_{\text{tot}} \boldsymbol{q} := P_f(\boldsymbol{q}); \{|-\theta\rangle\langle-\theta|\} \square \{|\theta+\delta\rangle\langle\theta+\delta|\} && \text{(H.unit)}\\
&\hookrightarrow_{\text{tot}} \boldsymbol{q} := P_f(\boldsymbol{q}); \boldsymbol{q} := (2|U\rangle\langle U| - I)(\boldsymbol{q}) && \text{(H.unit)}
\end{aligned}$$

The first application of (H.unit) is correct by our choice of intermediate condition, but for the second one we need that

$$(2|U\rangle\langle U| - I)\,|-\theta\rangle\langle-\theta|\,(2|U\rangle\langle U| - I) \Rightarrow |\theta+\delta\rangle\langle\theta+\delta|\,.$$

Since $|U\rangle$ is at angle v, reflecting about it sends $|-\theta\rangle = |v-(\theta+v)\rangle$ to $|v+(\theta+v)\rangle = |\theta+2v\rangle$. Thus the above refinements are valid if we choose $\delta = 2v$ as the rotation angle, where we recall that $v = \arcsin\sqrt{T/N}$. If we plug this back into Eq. (4.9) we find that the number of rotations is

$$r = \left[\frac{\frac{\pi}{2} - v}{2v}\right] = \left[\frac{\pi}{4\arcsin\sqrt{T/N}} - \frac{1}{2}\right] = O\left(\sqrt{\frac{N}{T}}\right).$$

Moreover, the success probability can be lower bounded as

$$p \geq \cos^2(\delta/2) = 1 - \sin^2(v) = 1 - T/N.$$

Thus we obtain the following quantum program:

$$S_{\text{search}}\left(1 - \frac{T}{N}\right) \hookrightarrow^*_{\text{tot}}$$

$$\begin{array}{l}
\boldsymbol{q} := |\boldsymbol{0}\rangle\,;\ \boldsymbol{q} := H^{\otimes n}(\boldsymbol{q});\\
\textbf{repeat}\ \left[\frac{\pi}{4\arcsin\sqrt{T/N}} - \frac{1}{2}\right]\ \textbf{do}\\
\quad \boldsymbol{q} := P_f(\boldsymbol{q});\ \boldsymbol{q} := (2|U\rangle\langle U| - I)(\boldsymbol{q})\\
\textbf{end}
\end{array}$$

As we have constructed it by refining the initial specification for the search problem, it satisfies the specification by construction. It succeeds with probability $p \geq 1 - T/N$ and uses $O(\sqrt{N/T})$ queries to the quantum oracle. This is in fact Grover's algorithm [30].

4.4 Boosting Success Probabilities

In this section, we derive two refinement rules that formalize useful and widely used patterns (see, e.g., [41, 52]). To motivate it, recall that in the preceding example, we constructed two quantum programs that succeed with some probability. We modeled this by a specification of the form $\{\varepsilon I\}\square\{Q\}$ for some $\varepsilon > 0$ (we now write ε rather than p because the discussion that follows is most relevant when ε is a small probability). Indeed, a Hoare triple $\{\varepsilon I\}S\{Q\}$ is totally correct if the program S terminates and the postcondition Q holds with probability at least ε. We can amplify or "boost" the success probability of such a program S arbitrarily by simply repeating it until the postcondition holds, provided (i) the program S terminates almost surely (so that we keep repeating) and (ii) the postcondition is given by a projection (so that measuring it does not impact its expectation). To incorporate the termination requirement we can consider the multiple specification $\{\varepsilon I, I\}\square\{Q, I\}$.

We first give a rule that reduces the construction of a program that succeeds with some probability $p \in (0,1)$ to the construction of a program that succeeds with some smaller probability $\varepsilon \in (0,p)$:

(H.boostRep) $\{pI, I\}\square\{Q_{\boldsymbol{q}}, I\} \hookrightarrow^*_{\text{tot}}$ **repeat** $\lceil \log_{1-\varepsilon}(1-p) \rceil$ **do** **if meas** $\boldsymbol{q}$ **with** $Q^\perp$ **then** $\{\varepsilon Q^\perp_{\boldsymbol{q}}, Q^\perp_{\boldsymbol{q}}\}\square\{Q_{\boldsymbol{q}}, I\}$ **end** **end**,

for any projection Q and any $\varepsilon \in (0,p)$, where $Q^\perp = I - Q$

Second, we give a rule to reduce the construction of programs that succeed with probability one to ones that succeed with some finite probability $\varepsilon \in (0,1)$, by repeating the program until it succeeds:

(H.boostWhile) $\{I\}\square\{Q_{\boldsymbol{q}}\} \hookrightarrow^*_{\text{tot}}$ **while meas** $\boldsymbol{q}$ **with** $Q^\perp$ **do** $\{\varepsilon Q^\perp_{\boldsymbol{q}}, Q^\perp_{\boldsymbol{q}}\}\square\{Q_{\boldsymbol{q}}, I\}$ **end**,

for any projection Q and any $\varepsilon \in (0,1)$, where $Q^\perp = I - Q$

Theorem 4.1 (Boosting success probability). *The relations (H.boostRep), (H.boostWhile) hold.*

The proof can be found in Appendix C. We emphasize that both rules are totally correct.

4.5 Quantum Fourier Transform

The Quantum Fourier Transform (QFT) is widely used in many algorithms, such as Shor's factoring algorithm [66] and quantum phase estimation [36]. For n qubits, it computes the following unitary:

$$\mathsf{QFT}_n = \frac{1}{\sqrt{2^n}} \sum_{x,y \in \{0,\ldots,2^n-1\}} \omega_n^{xy} \, |x\rangle\langle y|$$

where $\omega_n = \exp\left(\frac{2\pi i}{2^n}\right)$. Here, the n-bit numbers x, y are identified with big-endian bitstrings, that is, $|x\rangle = |x_1\rangle \otimes \ldots \otimes |x_n\rangle$ where $x = \sum_{j=1}^n x_j 2^{n-j}$. The key observation is that $\omega_k^2 = \omega_{k-1}$, which naturally motivates a recursive approach. Therefore, we will attempt to construct a program implementing QFT_n using a program for QFT_{n-1}, and so. In this section only, we will use the shorthand notation $|\Psi\rangle$ to represent the predicate $|\Psi\rangle\langle\Psi|$, which is a projection for any unit vector $|\Psi\rangle$.

Specification. As in the teleportation example (Section 4.2), we can fully specify the QFT by considering its action on half of a maximally entangled input state. To allow us to recurse on the number of qubits, we will define a specification for QFT_k for each k from 1 to n. To this end let S_k denote the specification for applying QFT_k on the first k qubits of an n-qubit quantum variable $\boldsymbol{q}$:

$$S_k = \{|\Phi_n^+\rangle_{\boldsymbol{r},\boldsymbol{q}}\}\square\{(\mathsf{QFT}_k)_{q_1,\ldots,q_k} \, |\Phi_n^+\rangle_{\boldsymbol{r},\boldsymbol{q}}\},$$

where $|\Phi_n^+\rangle_{\boldsymbol{r},\boldsymbol{q}} = \bigotimes_{j=1}^n |\phi^+\rangle_{r_j,q_j} = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle_{\boldsymbol{r}} \, |x\rangle_{\boldsymbol{q}}$ (with $|\phi^+\rangle$ as in Eq. (4.2)) is a maximally entangled state between $\boldsymbol{q}$ and an additional n-qubit variable $\boldsymbol{r}$ that will not be used in the program.

Any program obtained by refining S_k implements the QFT on the first k qubits of $\boldsymbol{q}$ while acting trivially on the rest. In particular, S_n specifies the n-qubit QFT.

Construction. We will show that a quantum program for S_n can be constructed recursively, starting with S_1 and subsequently constructing a program for S_k from one for S_{k-1} for any $k = 2, \dots, n$.

Base Case ($k = 1$). As $\omega_1 = -1$, we can see that $\mathsf{QFT}_1 = \frac{1}{\sqrt{2}} \sum_{x,y\in\{0,1\}} (-1)^{xy} |x\rangle\langle y| = H$ straight from the definition, i.e., the 1-qubit QFT is the Hadamard gate. Therefore, we can refine:

$$S_1 \hookrightarrow_{\text{tot}} \; q_1 := H(q_1) \tag{H.unit}$$

We can apply (H.unit) because $H_{q_1}^\dagger \left((\mathsf{QFT}_1)_{q_1} |\Phi_n^+\rangle\langle\Phi_n^+|_{\boldsymbol{r},\boldsymbol{q}} (\mathsf{QFT}_1)_{q_1}^\dagger \right) H_{q_1} = |\Phi_n^+\rangle\langle\Phi_n^+|_{\boldsymbol{r},\boldsymbol{q}}$ holds.

General Case ($k > 1$). We refine S_k by using (H.seq) into a sequence of two holes, with intermediate condition being the postcondition of S_{k-1}. Then the first hole in the sequence matches S_{k-1}:

$$S_k \hookrightarrow_{\text{tot}} \; S_{k-1}\,; \{(\mathsf{QFT}_{k-1})_{q_1,\dots,q_{k-1}} |\Phi_n^+\rangle_{\boldsymbol{r},\boldsymbol{q}}\} \square \{(\mathsf{QFT}_k)_{q_1,\dots,q_k} |\Phi_n^+\rangle_{\boldsymbol{r},\boldsymbol{q}}\} \tag{H.seq}$$

The specification S_{k-1} can be refined recursively, but we still need to refine the right-hand side hole. Let us denote the states defining the pre- and postcondition by $|\Psi_{k-1}\rangle := (\mathsf{QFT}_{k-1})_{q_1,\dots,q_{k-1}} |\Phi_n^+\rangle_{\boldsymbol{r},\boldsymbol{q}}$ and $|\Psi_k\rangle := (\mathsf{QFT}_k)_{q_1,\dots,q_k} |\Phi_n^+\rangle_{\boldsymbol{r},\boldsymbol{q}}$, respectively. Because the program should only act on the first k qubits, we expand these accordingly:

$$|\Psi_{k-1}\rangle = \left(\frac{1}{2^{k-1}\sqrt{2}} \sum_{\tilde{x},\tilde{y}=0}^{2^{k-1}-1} \sum_{x_k\in\{0,1\}} \omega_{k-1}^{\tilde{x}\tilde{y}} |\tilde{x}\rangle_{r_1,\dots,r_{k-1}} |x_k\rangle_{r_k} |\tilde{y}\rangle_{q_1,\dots,q_{k-1}} |x_k\rangle_{q_k} \right) \otimes |\Psi^{\text{rest}}\rangle\,,$$

$$|\Psi_k\rangle = \left(\frac{1}{2^k} \sum_{x,y=0}^{2^k-1} \omega_k^{xy} |x\rangle_{r_1,\dots,r_k} |y\rangle_{q_1,\dots,q_k} \right) \otimes |\Psi^{\text{rest}}\rangle\,,$$

where $|\Psi^{\text{rest}}\rangle$ is the (unchanged) state on the last $n-k$ qubits of $\boldsymbol{r},\boldsymbol{q}$. To compare the state $|\Psi_k\rangle$ with $|\Psi_{k-1}\rangle$, let us write $x = 2\tilde{x} + x_k$. Then we can also simplify

$$\omega_k^{xy} = \omega_k^{2\tilde{x}y} \omega_k^{x_k y} = \omega_{k-1}^{\tilde{x}y} \omega_k^{yx_k}.$$

Now as $\omega_{k-1}^{2^{k-1}} = 1$, the most significant bit of y does not affect the first part of the above term. Therefore, it is natural to write the index y as $y = y_1 2^{k-1} + \tilde{y}$, so that we can further simplify

$$\omega_{k-1}^{\tilde{x}y} \omega_k^{yx_k} = \omega_{k-1}^{\tilde{x}\tilde{y}} (-1)^{y_1 x_k} \omega_k^{\tilde{y}x_k}.$$

Altogether we find that the postcondition is described by the state

$$|\Psi_k\rangle = \left(\frac{1}{2^k} \sum_{\tilde{x},\tilde{y}=0}^{2^{k-1}-1} \sum_{x_k,y_1\in\{0,1\}} \omega_{k-1}^{\tilde{x}\tilde{y}} (-1)^{y_1 x_k} \omega_k^{\tilde{y}x_k} |\tilde{x}\rangle_{r_1,\dots,r_{k-1}} |x_k\rangle_{r_k} |y_1\rangle_{q_1} |\tilde{y}\rangle_{q_2,\dots,q_k} \right) \otimes |\Psi^{\text{rest}}\rangle\,.$$

The state $|\Psi_k\rangle$ looks quite similar to $|\Psi_{k-1}\rangle$ but there are some key differences. For one, the index $\tilde{y}$ refers qubits $q_2, \dots, q_k$ rather than $q_1, \dots, q_{k-1}$ and hence q_1 plays a distinguished role rather than q_k. Thus a natural first step is to move the k-th qubit to the front by applying a suitable sequence of swaps. That is, we use (H.seq) and (H.unit) to refine

$$\{|\Psi_{k-1}\rangle\} \square \{\Psi_k\} \hookrightarrow_{\text{tot}}^* \; q_k, q_{k-1} := \mathsf{SWAP}(q_k, q_{k-1}); \dots; q_2, q_1 := \mathsf{SWAP}(q_2, q_1); \{|\Psi'\rangle\} \square \{|\Psi_k\rangle\}\,,$$

which is allowed if we pick the intermediate condition given by the state

$$|\Psi'\rangle = \left(\frac{1}{2^{k-1}\sqrt{2}} \sum_{\tilde{x},\tilde{y}=0}^{2^{k-1}-1} \sum_{x_k \in \{0,1\}} \omega_{k-1}^{\tilde{x}\tilde{y}} |\tilde{x}\rangle_{r_1,\ldots,r_{k-1}} |x_k\rangle_{r_k} |x_k\rangle_{q_1} |\tilde{y}\rangle_{q_2,\ldots,q_k} \right) \otimes |\Psi^{\text{rest}}\rangle .$$

Now the qubits are in the right place but we see that in $|\Psi_k\rangle$ there is an additional relative phase $\omega_k^{\tilde{y}x_k}$. Because $\omega_k^{\tilde{y}x_k} = (\omega_k^{y_k} \omega_{k-1}^{y_{k-1}} \ldots \omega_2^{y_2})^{x_k}$ we are led to refining the second hole by a series of phase gates, one on each qubit $q_2, \ldots, q_k$, and each controlled by the value x_k that is stored in qubit q_1:

$$\{|\Psi'\rangle\} \square \{|\Psi_k\rangle\} \hookrightarrow_{\text{tot}}^{*} q_1, q_2 := \mathsf{CRz}_2(q_1, q_2); \ldots; q_1, q_k := \mathsf{CRz}_k(q_1, q_k); \{|\Psi''\rangle\} \square \{|\Psi_k\rangle\}$$

where $\mathsf{Rz}_k = \left(\begin{smallmatrix} 1 & 0 \\ 0 & \omega_k \end{smallmatrix}\right)$ and CRz_k denotes the corresponding controlled gate. The above refinement is valid if pick the intermediate condition given by the state

$$|\Psi''\rangle = \left(\frac{1}{2^{k-1}\sqrt{2}} \sum_{\tilde{x},\tilde{y}=0}^{2^{k-1}-1} \sum_{x_k \in \{0,1\}} \omega_{k-1}^{\tilde{x}\tilde{y}} \omega_k^{\tilde{y}x_k} |\tilde{x}\rangle_{r_1,\ldots,r_{k-1}} |x_k\rangle_{r_k} |x_k\rangle_{q_1} |\tilde{y}\rangle_{q_2,\ldots,q_k} \right) \otimes |\Psi^{\text{rest}}\rangle$$

This is almost identical to $|\Psi_k\rangle$, except that q_1 is in a basis state rather than a suitable superposition. As we have $H|x_k\rangle = \frac{1}{\sqrt{2}} \sum_{y_1} (-1)^{x_k y_1} |y_1\rangle$ by definition, this is easily fixed by a Hadamard gate:

$$\{|\Psi''\rangle\} \square \{|\Psi_k\rangle\} \hookrightarrow_{\text{tot}} q_1 := H(q_1) . \tag{H.unit}$$

This concludes the construction of the quantum Fourier transform.

5 Conclusion and Outlook

In this work, we proposed Quantum Correctness by Construction (QbC), an approach for constructing quantum programs that are guaranteed to be correct by construction. To this end, we extended a quantum while language with a construct called *holes*, which represent yet-to-be-constructed subprograms that carry a precondition and a postcondition. We presented refinement rules that iteratively refine such quantum programs and proved that these rules are sound and complete: every program is guaranteed to satisfy the specification it was constructed from, and every correct program can always be constructed from the specification. Finally, we demonstrated the QbC approach by constructing quantum programs for some idiomatic problems, starting from their natural specification. We found that in these examples, QbC naturally suggested how to derive program details and highlighted key design choices that had to be made along the way. We take these findings to suggest that QbC could play a meaningful role in supporting the design of quantum algorithms, their taxonomization, and the construction and verification of larger quantum software. We now describe some promising directions for future research to further pave the way in this direction and conclude with a perspective on the role of automation in algorithm development.

Future Directions: Theory. A natural and interesting direction would be to extend the QbC methodology to other settings and non-functional properties, such as by building on the expected-runtime calculus introduced by Liu et al. [42] to construct programs that are *efficient by construction*. Another direction would be to extend the QbC approach to other quantum programming languages. While the *quantum while language* used in our paper is well-understood to provide a clean theoretical model, it is often cumbersome to express complex quantum programs in it. It would therefore be desirable to extend QbC to a more expressive language, which might include both classical and quantum variables [24], oracles and subroutines, quantum data structures, and so forth. It would also be highly interesting to identify further refinement rules that encode high-level reasoning and design patterns that are commonly used in quantum algorithms, and extend the language to natively support these operations. For example, quantum amplitude amplification [16], which

generalizes Grover's algorithm to offer a quantum speedup for boosting the success probability of a subroutine that improves over naive repetition (cf. Sections 4.3 and 4.4) and is widely used. It would also be interesting to devise hybrid approaches that combine both post-hoc (Hoare or weakest-precondition logic) and by-construction (QbC) reasoning, which can be useful particularly when constructing larger and more complex programs [75].

Future Directions: Implementation and Mechanization. Our framework and results are agnostic to the choice of assertion language, in the interest of generality. But committing to a concrete assertion language is an important choice for implementations. From our examples and refinement rules, we find that it is convenient to consider projections scaled by scalar values and finite linear combinations thereof, expressed in Dirac notation. For instance, a web-based prototype proposed recently uses a simple assertion language based on Dirac notation for a fixed number of qubits [65]. The conditions of refinement are checked using a decision procedure on finite-sized complex matrices to verify the Löwner order. To further mechanize QbC and in particular to handle general programs when the number of qubits is not fixed, we believe a natural and ambitious future work would be to integrate QbC with a proof assistant such as Coq or Lean to handle proof obligations of side conditions. Notable prior work includes CoqQ [82] and QWIRE [57], which formalize reasoning about quantum programs in CoQ, and the recent decision procedure for Dirac notation [77].

Outlook: Algorithm Development. Our view is that algorithm design is a creative process that often requires insights that are difficult to obtain purely by automation. We believe that a by-construction approach can help algorithm designers focus on this creative aspect, one key insight at a time, to develop algorithms that are ensured to be formally correct, without being bogged down by small details, as the framework guides the refinement and gives a principled way to generate side-conditions that need to be checked per refinement step. Several well-known quantum algorithms have been post-hoc verified using the quantum Hoare logic proof system. It is also possible to construct these same programs using QbC, at roughly the same complexity. Compared with post-hoc verification, this can provide additional benefits: one does not have to decide on all program details a priori, but can do so during the refinement process. This can lead to different choices, resulting in different programs and trade-offs. We showcased this in the search example where we derived two programs from the same specification, by making a different choice at a key step. Similar examples are known in the classical CbC literature. This may also open up the possibility of discovering alternate and cleaner implementations to existing algorithms. However, we emphasize that we do not see by-construction and post-hoc approaches as mutually exclusive, but rather as complementing each other with each having its role. The most natural approach may well be a hybrid approach, as we propose to explore in future work above.

6 Data-Availability Statement

This paper proposes the theoretical foundations for a correctness-by-construction approach for quantum programs. We do not provide an artifact, but note that [65] gives a web-based prototype.

Acknowledgments

We thank Gilles Barthe and Bruce Watson for fruitful discussions on the subject of this work, and the anonymous referees for valuable feedback on earlier versions of this manuscript. All authors acknowledge support by the BMBF (QuBRA, 13N16135 & 13N16303; QuSol, 13N17173 & 13N17170). IS also acknowledges support by the BMWK (ProvideQ, 01MQ22006F). MW also acknowledges support by the European Union (ERC, SYMOPTIC, 101040907), by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972, and by the Dutch Research Council (NWO grant OCENW.KLEIN.267).

References

[1] J.-R. Abrial. 1996. *The B-Book: Assigning Programs to Meanings.* Cambridge University Press, USA.

[2] Jean-Raymond Abrial. 2010. *Modeling in Event-B: System and Software Engineering* (1st ed.). Cambridge University Press, USA.

[3] T. Altenkirch and J. Grattage. 2005. A functional quantum programming language. In *20th Annual IEEE Symposium on Logic in Computer Science (LICS' 05)*. IEEE, New York, NY, USA, 249–258. doi:10.1109/LICS.2005.1

[4] Andris Ambainis. 2004. Quantum search algorithms. *ACM SIGACT News* 35, 2 (2004), 22–35.

[5] Roman Andriushchenko, Milan Češka, Sebastian Junges, Joost-Pieter Katoen, and Šimon Stupinský. 2021. PAYNT: A Tool for Inductive Synthesis of Probabilistic Programs. In *Computer Aided Verification: 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021, Proceedings, Part I*. Springer-Verlag, Berlin, Heidelberg, 856–869. doi:10.1007/978-3-030-81685-8_40

[6] Simon Apers, Stacey Jeffery, Galina Pass, and Michael Walter. 2023. (No) Quantum Space-Time Tradeoff for USTCON. In *31st Annual European Symposium on Algorithms (ESA 2023) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 274)*, Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 10:1–10:17. doi:10.4230/LIPIcs.ESA.2023.10

[7] Krzysztof R. Apt and Ernst-Rüdiger Olderog. 2019. Fifty years of Hoare's logic. *Formal Aspects of Computing* 31, 6 (Dec. 2019), 751–807. doi:10.1007/s00165-019-00501-3

[8] Martin Avanzini, Gilles Barthe, Davide Davoli, and Benjamin Grégoire. 2025. A Quantitative Probabilistic Relational Hoare Logic. *Proc. ACM Program. Lang.* 9, POPL, Article 40 (Jan. 2025), 29 pages. doi:10.1145/3704876

[9] Ralph-Johan J. Back, Abo Akademi, J. Von Wright, F. B. Schneider, and D. Gries. 1998. *Refinement Calculus: A Systematic Introduction* (1st ed.). Springer-Verlag, Berlin, Heidelberg.

[10] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. 2009. Formal certification of code-based cryptographic proofs. In *Proceedings of the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Savannah, GA, USA) *(POPL '09)*. Association for Computing Machinery, New York, NY, USA, 90–101. doi:10.1145/1480881.1480894

[11] Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Thomas Noll. 2019. Quantitative Separation Logic: A Logic for Reasoning about Probabilistic Pointer Programs. *Proc. ACM Program. Lang.* 3, POPL, Article 34 (jan 2019), 29 pages. doi:10.1145/3290347

[12] 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. doi:10.1145/3385412.3386007

[13] Richard Bird and Oege de Moor. 1997. *Algebra of programming.* Prentice-Hall, Inc., USA.

[14] Tabea Bordis, Loek Cleophas, Alexander Kittelmann, Tobias Runge, Ina Schaefer, and Bruce W. Watson. 2022. ReCorC-ing KeY: Correct-by-Construction Software Development Based on KeY. In *The Logic of Software. A Tasting Menu of Formal Methods (Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics))*, Wolfgang Ahrendt, Bernhard Beckert, Richard Bubel, and Einar Broch Johnsen (Eds.). Springer, Germany, 80–104. doi:10.1007/978-3-031-08166-8_5 Publisher Copyright: © 2022, Springer Nature Switzerland AG..

[15] Tabea Bordis, Tobias Runge, Alexander Kittelmann, and Ina Schaefer. 2023. Correctness-by-Construction: An Overview of the CorC Ecosystem. *Ada Lett.* 42, 2 (apr 2023), 75–78. doi:10.1145/3591335.3591343

[16] Gilles Brassard, Peter Hoyer, Michele Mosca, and Alain Tapp. 2002. Quantum amplitude amplification and estimation. *Contemp. Math.* 305 (2002), 53–74.

[17] Dominique Cansell and Dominique Mery. 2003. Foundations of the B Method. *Computers and Informatics* 22 (01 2003), 31 p.

[18] R. Chadha, P. Mateus, and A. Sernadas. 2006. Reasoning About Imperative Quantum Programs. *Electron. Notes Theor. Comput. Sci.* 158 (may 2006), 19–39. doi:10.1016/j.entcs.2006.04.003

[19] Shantanav Chakraborty, Aditya Morolia, and Anurudh Peduri. 2023. Quantum Regularized Least Squares. *Quantum* 7 (April 2023), 988. doi:10.22331/q-2023-04-27-988

[20] Andrew M. Childs, Richard Cleve, Enrico Deotto, Edward Farhi, Sam Gutmann, and Daniel A. Spielman. 2003. Exponential Algorithmic Speedup by a Quantum Walk. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing* (San Diego, CA, USA) *(STOC '03)*. Association for Computing Machinery, New York, NY, USA, 59–68. doi:10.1145/780542.780552

[21] Adam Chlipala. 2013. *Certified programming with dependent types.* MIT Press, London, England.

[22] J. I. den Hartog. 1999. Verifying Probabilistic Programs Using a Hoare like Logic. In *Advances in Computing Science — ASIAN'99*, P. S. Thiagarajan and Roland Yap (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 113–125.

[23] Edsger Wybe Dijkstra. 1997. *A Discipline of Programming* (1st ed.). Prentice Hall PTR, USA.

[24] Yuan Feng and Mingsheng Ying. 2021. Quantum Hoare logic with classical variables. *ACM Transactions on Quantum Computing* 2, 4 (2021), 1–43.

[25] Yuan Feng, Li Zhou, and Yingte Xu. 2023. Refinement calculus of quantum programs with projective assertions. arXiv:2311.14215 [cs.LO]

[26] András Gilyén, Yuan Su, Guang Hao Low, and Nathan Wiebe. 2019. Quantum Singular Value Transformation and beyond: Exponential Improvements for Quantum Matrix Arithmetics. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing* (Phoenix, AZ, USA) *(STOC 2019)*. Association for Computing Machinery, New York, NY, USA, 193–204. doi:10.1145/3313276.3316366

[27] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. 2008. Quantum Random Access Memory. *Phys. Rev. Lett.* 100 (Apr 2008), 160501. Issue 16. doi:10.1103/PhysRevLett.100.160501

[28] Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. 2013. Quipper: A Scalable Quantum Programming Language. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Seattle, Washington, USA) *(PLDI '13)*. Association for Computing Machinery, New York, NY, USA, 333–342. doi:10.1145/2491956.2462177

[29] David Gries. 1981. *The Science of Programming*. Springer New York, New York, NY. doi:10.1007/978-1-4612-5983-1

[30] Lov K. Grover. 1996. A Fast Quantum Mechanical Algorithm for Database Search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing* (Philadelphia, Pennsylvania, USA) *(STOC '96)*. Association for Computing Machinery, New York, NY, USA, 212–219. doi:10.1145/237814.237866

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

[32] HaskellWiki. 2014. GHC/Typed holes — HaskellWiki. https://wiki.haskell.org/index.php?title=GHC/Typed_holes&oldid=58717 [Online; accessed 25-January-2024].

[33] C. A. R. Hoare. 1969. An Axiomatic Basis for Computer Programming. *Commun. ACM* 12, 10 (oct 1969), 576–580. doi:10.1145/363235.363259

[34] Yoshihiko Kakutani. 2009. A Logic for Formal Verification of Quantum Programs. In *Advances in Computer Science - ASIAN 2009. Information Security and Privacy*, Anupam Datta (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 79–93.

[35] Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo. 2016. Weakest precondition reasoning for expected run–times of probabilistic programs. In *Programming Languages and Systems: 25th European Symposium on Programming, ESOP 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2–8, 2016, Proceedings 25*, Peter Thiemann (Ed.). Springer, Springer Berlin Heidelberg, Berlin, Heidelberg, 364–389.

[36] A. Yu. Kitaev. 1995. Quantum measurements and the Abelian Stabilizer Problem. arXiv:quant-ph/9511026 [quant-ph] https://arxiv.org/abs/quant-ph/9511026

[37] Alexander Knüppel, Tobias Runge, and Ina Schaefer. 2020. Scaling Correctness-by-Construction. In *Leveraging Applications of Formal Methods, Verification and Validation: Verification Principles: 9th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2020, Rhodes, Greece, October 20-30, 2020, Proceedings, Part I* (Rhodes, Greece). Springer-Verlag, Berlin, Heidelberg, 187–207. doi:10.1007/978-3-030-61362-4_10

[38] Derrick G. Kourie and Bruce W. Watson. 2012. *The Correctness-by-Construction Approach to Programming*. Springer, Berlin, Heidelberg. doi:10.1007/978-3-642-27919-5

[39] 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

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

[41] Yuan Liang Lim, Almut Beige, and Leong Chuan Kwek. 2005. Repeat-Until-Success Linear Optics Distributed Quantum Computing. *Phys. Rev. Lett.* 95 (Jul 2005), 030505. Issue 3. doi:10.1103/PhysRevLett.95.030505

[42] Junyi Liu, Li Zhou, Gilles Barthe, and Mingsheng Ying. 2022. Quantum Weakest Preconditions for Reasoning about Expected Runtimes of Quantum Programs. In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science* (Haifa, Israel) *(LICS '22)*. Association for Computing Machinery, New York, NY, USA, Article 4, 13 pages. doi:10.1145/3531130.3533327

[43] John M. Martyn, Zane M. Rossi, Andrew K. Tan, and Isaac L. Chuang. 2021. Grand Unification of Quantum Algorithms. *PRX Quantum* 2 (Dec 2021), 040203. Issue 4. doi:10.1103/PRXQuantum.2.040203

[44] Ashley Montanaro. 2016. Quantum algorithms: an overview. *npj Quantum Information* 2, 1 (Jan. 2016), 1–8. doi:10.1038/npjqi.2015.23 Number: 1 Publisher: Nature Publishing Group.

[45] Carroll Morgan. 1988. The specification statement. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 10, 3 (1988), 403–419.

[46] Carroll Morgan and Annabelle McIver. 1999. pGCL: Formal reasoning for random algorithms. *South African Computer Journal*, 14–27.

[47] Ana Neri, Rui Soares Barbosa, and José N Oliveira. 2021. Compiling quantamorphisms for the IBM Q Experience. *IEEE Transactions on Software Engineering* 48, 11 (2021), 4339–4356.

[48] Michael A Nielsen and Isaac L Chuang. 2010. *Quantum Computation and Quantum Information.* Cambridge University Press, Cambridge. doi:10.1017/CBO9780511976667

[49] Peter W. O'Hearn. 2019. Incorrectness Logic. *Proc. ACM Program. Lang.* 4, POPL, Article 10 (dec 2019), 32 pages. doi:10.1145/3371078

[50] Federico Olmedo, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. 2016. Reasoning about Recursive Probabilistic Programs. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science* (New York, NY, USA) *(LICS '16)*. Association for Computing Machinery, New York, NY, USA, 672–681. doi:10.1145/2933575.2935317

[51] Cyrus Omar, Ian Voysey, Ravi Chugh, and Matthew A. Hammer. 2019. Live functional programming with typed holes. *Proc. ACM Program. Lang.* 3, POPL, Article 14 (jan 2019), 32 pages. doi:10.1145/3290327

[52] Adam Paetznick and Krysta M. Svore. 2014. Repeat-until-success: non-deterministic decomposition of single-qubit unitaries. *Quantum Info. Comput.* 14, 15-16 (nov 2014), 1277–1301.

[53] Raúl Pardo, Einar Broch Johnsen, Ina Schaefer, and Andrzej Wąsowski. 2022. A Specification Logic for Programs in the Probabilistic Guarded Command Language. In *Theoretical Aspects of Computing – ICTAC 2022*, Helmut Seidl, Zhiming Liu, and Corina S. Pasareanu (Eds.). Springer International Publishing, Cham, 369–387.

[54] 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* (Paris, France) *(POPL '17)*. Association for Computing Machinery, New York, NY, USA, 846–858. doi:10.1145/3009837.3009894

[55] Robert Rand. 2019. Verification Logics for Quantum Programs. arXiv:1904.04304 [cs.LO]

[56] Robert Rand, Jennifer Paykin, Dong-Ho Lee, and Steve Zdancewic. 2019. ReQWIRE: Reasoning about Reversible Quantum Circuits. *Electronic Proceedings in Theoretical Computer Science* 287 (jan 2019), 299–312. doi:10.4204/eptcs.287.17

[57] Robert Rand, Jennifer Paykin, and Steve Zdancewic. 2018. QWIRE Practice: Formal Verification of Quantum Circuits in Coq. *Electronic Proceedings in Theoretical Computer Science* 266 (feb 2018), 119–132. doi:10.4204/eptcs.266.8

[58] Tobias Runge, Tabea Bordis, Alex Potanin, Thomas Thüm, and Ina Schaefer. 2023. Flexible Correct-by-Construction Programming. *Logical Methods in Computer Science* Volume 19, Issue 2 (June 2023). doi:10.46298/lmcs-19(2:16)2023

[59] Tobias Runge, Alexander Knüppel, Thomas Thüm, and Ina Schaefer. 2020. Lattice-Based Information Flow Control-by-Construction for Security-by-Design. In *Proceedings of the 8th International Conference on Formal Methods in Software Engineering* (Seoul, Republic of Korea) *(FormaliSE '20)*. Association for Computing Machinery, New York, NY, USA, 44–54. doi:10.1145/3372020.3391565

[60] Tobias Runge, Alex Potanin, Thomas Thüm, and Ina Schaefer. 2022. Traits: Correctness-by-Construction For Free. In *Formal Techniques for Distributed Objects, Components, and Systems: 42nd IFIP WG 6.1 International Conference, FORTE 2022, Held as Part of the 17th International Federated Conference on Distributed Computing Techniques, DisCoTec 2022, Lucca, Italy, June 13-17, 2022, Proceedings* (Lucca, Italy). Springer-Verlag, Berlin, Heidelberg, 131–150. doi:10.1007/978-3-031-08679-3_9

[61] Tobias Runge, Ina Schaefer, Loek Cleophas, Thomas Thüm, Derrick Kourie, and Bruce W. Watson. 2019. Tool Support for Correctness-by-Construction. In *Fundamental Approaches to Software Engineering*, Reiner Hähnle and Wil van der Aalst (Eds.). Springer International Publishing, Cham, 25–42.

[62] Tobias Runge, Thomas Thüm, Loek Cleophas, Ina Schaefer, and Bruce W. Watson. 2020. Comparing Correctness-by-Construction with Post-Hoc Verification—A Qualitative User Study. In *Formal Methods. FM 2019 International Workshops*, Emil Sekerinski, Nelma Moreira, José N. Oliveira, Daniel Ratiu, Riccardo Guidotti, Marie Farrell, Matt Luckcuck, Diego Marmsoler, José Campos, Troy Astarte, Laure Gonnord, Antonio Cerone, Luis Couto, Brijesh Dongol, Martin Kutrib, Pedro Monteiro, and David Delmas (Eds.). Springer International Publishing, Cham, 388–405.

[63] Jeff W Sanders and Paolo Zuliani. 2000. Quantum programming. In *International Conference on Mathematics of Program Construction*. Springer, 80–99.

[64] Peter Selinger. 2004. Towards a quantum programming language. *Mathematical Structures in Computer Science* 14, 4 (2004), 527–586. doi:10.1017/S0960129504004256

[65] Niklas Seng. 2024. *Quantum Correctness By Construction On The Web.* Bachelor's thesis. Karlsruhe Institute of Technology. See http://qbc.kastel.kit.edu and also http://qbc.kastel.kit.edu/tutorial.

[66] P.W. Shor. 1994. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*. 124–134. doi:10.1109/SFCS.1994.365700

[67] Jeremy Siek and Walid Taha. 2006. Gradual typing for functional languages. *Scheme and Functional Programming*.
[68] Armando Solar-Lezama. 2009. The sketching approach to program synthesis. In *Asian Symposium on Programming Languages and Systems*. Springer, 4–13.
[69] Damian S. Steiger, Thomas Häner, and Matthias Troyer. 2018. ProjectQ: an open source software framework for quantum computing. *Quantum* 2 (Jan. 2018), 49. doi:10.22331/q-2018-01-31-49
[70] Dominic Steinhöfel and Reiner Hähnle. 2019. Abstract Execution. In *Formal Methods – The Next 30 Years*, Maurice H. ter Beek, Annabelle McIver, and José N. Oliveira (Eds.). Springer International Publishing, Cham, 319–336.
[71] 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* (Vienna, Austria) *(RWDSL2018)*. Association for Computing Machinery, New York, NY, USA, Article 7, 10 pages. doi:10.1145/3183895.3183901
[72] Dominique Unruh. 2019. Quantum relational Hoare logic. *Proceedings of the ACM on Programming Languages* 3, POPL (Jan. 2019), 1–31. doi:10.1145/3290346
[73] Joran van Apeldoorn, Sander Gribling, Yinan Li, Harold Nieuwboer, Michael Walter, and Ronald de Wolf. 2021. Quantum Algorithms for Matrix Scaling and Matrix Balancing. In *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 198)*, Nikhil Bansal, Emanuela Merelli, and James Worrell (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 110:1–110:17. doi:10.4230/LIPIcs.ICALP.2021.110
[74] 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. doi:10.1145/3571225
[75] Bruce W. Watson, Derrick G. Kourie, Ina Schaefer, and Loek Cleophas. 2016. Correctness-by-Construction and Post-hoc Verification: A Marriage of Convenience?. In *Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques*, Tiziana Margaria and Bernhard Steffen (Eds.). Springer International Publishing, Cham, 730–748.
[76] Mark M Wilde. 2013. *Quantum information theory*. Cambridge University Press, Cambridge. doi:10.1017/CBO9781139525343
[77] Yingte Xu, Gilles Barthe, and Li Zhou. 2025. Automating Equational Proofs in Dirac Notation. *Proc. ACM Program. Lang.* 9, POPL, Article 42 (Jan. 2025), 33 pages. doi:10.1145/3704878
[78] Peng Yan, Hanru Jiang, and Nengkun Yu. 2022. On Incorrectness Logic for Quantum Programs. *Proc. ACM Program. Lang.* 6, OOPSLA1, Article 72 (apr 2022), 28 pages. doi:10.1145/3527316
[79] N.S. Yanofsky and M.A. Mannucci. 2008. *Quantum Computing for Computer Scientists*. Cambridge University Press. doi:10.1017/CBO9780511813887
[80] Mingsheng Ying. 2012. Floyd–Hoare Logic for Quantum Programs. *ACM Trans. Program. Lang. Syst.* 33, 6, Article 19 (2012), 49 pages. doi:10.1145/2049706.2049708
[81] Charles Yuan and Michael Carbin. 2022. Tower: Data Structures in Quantum Superposition. *Proc. ACM Program. Lang.* 6, OOPSLA2, Article 134 (oct 2022), 30 pages. doi:10.1145/3563297
[82] 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. doi:10.1145/3571222
[83] Li Zhou, Nengkun Yu, and Mingsheng Ying. 2019. An Applied Quantum Hoare Logic. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Phoenix, AZ, USA) *(PLDI 2019)*. Association for Computing Machinery, New York, NY, USA, 1149–1162. doi:10.1145/3314221.3314584
[84] Paolo Zuliani. 2007. A Formal Derivation of Grover's Quantum Search Algorithm. In *First Joint IEEE/IFIP Symposium on Theoretical Aspects of Software Engineering (TASE '07)*. 67–74. doi:10.1109/TASE.2007.3

Received 2024-10-15; accepted 2025-02-18